2003

Adaptive fault diagnosis in interactive electronic technical manuals (IETMs)

Rabih Faysal Kraidli
West Virginia University

Adaptive Fault Diagnosis in Interactive Electronic Technical Manuals (IETMs)

Rabih F. Kraidli

Thesis Submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science

Hany H. Ammar, Ph.D., Chair

Katerina Goseva-Popstojanova, Ph.D.

Don Reynolds, Technical Director, ManTech Enterprise Integration Center

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia University

2003

Keywords: IETM, Web Services, Adaptive Fault Diagnosis, User Modeling

ABSTRACT

Adaptive Fault Diagnosis in Interactive Electronic Technical Manuals (IETMs)

Rabih F. Kraidli

An Interactive Electronic Technical Manual (IETM) is a technical manual that is prepared in digital format to provide information about the diagnostics and maintenance of complex systems. Standards are issued by several organizations that dictate how such manuals are to be organized, primarily written in SGML. An IETM, by its nature, contains huge amounts of complex data and can run to be overwhelming to some users, hence decreasing their efficiency levels in using the manual and performing the task they are trying to look up. One type of such data is fault diagnosis, which is information pertaining to finding the cause of a specific fault that can occur in the system. Hence, we would like an IETM to be adapted to individuals according to their level of expertise, making the IETM generally, and performing a specific fault procedure specifically, easier for non experienced users. However, no current IETM standard dictates how this information should be adapted to specific classes of users.

One such standard is S1000D, issued by the European Association of Aerospace Industries (AECMA). S1000D is a simple standard that contains mainly information about how an IETM should be structured. This simplicity made it popular among the IETM community .However, S1000D doesn't contain information about how such data should be adapted. In this thesis we present a methodology for adaptive fault diagnosis in IETMs, a methodology that constantly adapts the fault diagnosis procedure, according to the experience of the user performing the diagnosis. We develop a framework of adaptation that constantly monitors user behavior, and "learns" about the fault and its possible causes as the system is used, hence making it easier to perform such procedures, which increases efficiency of usage of such a manual, an essential factor in performing fault diagnosis. We will also extend S1000D to incorporate all information necessary for our adaptation methodology.

The outcome of our methodology will be an IETM which contains "adaptable" fault diagnosis procedures that adapt to users according to their expertise levels making these procedures less cumbersome for users to accomplish, hence increasing their productivity and efficiency.

ACKNOWLEDGMENTS

First, I would like to express my deepest gratitude and appreciation to my research advisor, Dr. Hany Ammar, for this opportunity he gave me to conduct research under his supervision, and his ever presence guidance during this research effort.

I would also like to thank Dr. Nashaat Mansour of the Lebanese American University, who guided me throughout my undergraduate studies, and provided me with the opportunity to conduct research under Dr. Hany Ammar.

I would like to thank Dr. Katerina Goseva-Popstojanova for her support and review and for serving as a member in my graduate committee.

I would like to thank Mr. Don Reynolds, Technical Director, ManTech Enterprise Integration Center, for his time for serving on my committee and for his valuable feed back on this research effort, as well his ever present support and encouragement.

I would also like to thank Mr. Glen Copen, of ManTech Inc, for his time and support during the duration of this project.

Most of all I would like to thank my father for always motivating me to pursue the "better" in life, and for teaching me how to reach for the stars and never settle for the moon. Without him there is no me.

I would like to thank my mother for always being there for me, no matter what the circumstances, I am in her debt forever.

I would like to thank my loving sisters: Jommana, Hala and Ghada for providing all the support they did, and for making my stay in the US that much easier. I would also like to thank my brothers: Ghazi and Ziad, for being the best brothers any one could ever dream of. I would especially like to thank my brother Ghassan, for being so supportive over the years and over the troubling times and for always providing the light that shines my way when all lights seem to go out, thank you very much.

Last but not least, I would like to thank my research colleagues, especially Dr. Ahmad Hassan, for the guidance and expertise he provided through out this research effort.

This work was funded by ManTech Advanced Systems International, by a research grant to West Virginia University through the Software Engineering Research Center (SERC).

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 What is an IETM?

An Interactive Electronic Technical Manual (IETM) [1] is a Technical Manual, meaning that its primary purpose is to provide support to the diagnostics, maintenance, and repair of complex technical systems of military or commercial nature. These manuals contain descriptive, procedural, diagnostic as well as parts data, related to the system being described in the manual.

As its name implies, an IETM is prepared electronically, i.e. in digital format. It is prepared on a suitable medium with the aid of an automated authoring system, and designed to be displayed on an electronic screen to the end user. An IETM is also interactive; it interacts with its user, and dynamically presents data according to the user's inputs.

Many organizations have issued standards that dictate how such manuals are to be authored, structured, acquired and displayed. One of the forerunners in this is the U.S. Department of Defense (DOD) [2], which in 1992, issued military specifications for service-wide use in the acquisition of IETMs. On the other side of the Atlantic, the European Association of Aerospace Industries (AECMA) [3] also issued several specifications for IETMs, namely specification S1000D [4].

However as individual IETM implementations of the above mentioned standards have matured, the issue of interactivity in the IETM came out to be the major obstacle facing the development of such systems.

1.2 IETM Interactivity Issues

IETM specifications usually make a separation between the core database information and the presentation attributes of the system. The primary focus of the standards of

IETMs has been primarily on the core database part rather than on how the manual should be displayed.

The core database information part of the Interactive Electronic Technical Manual (IETM) consists of information regarding the system that the manual is documenting. These systems tend to be complex in nature, and hence the amount of information present in the manual can run to be overwhelming to the user. Hence, IETM specifications have tried to provide interactivity in the manual to prevent the IETM from being too complex for a user to use productively.

In the context of Interactive Electronic Technical Manuals (IETMs), "Interactive" means that a software application implementing the manual reacts to input from users as it is received. This reaction is often to tailor the content and presentation of subsequent display of information. The DoD IETM specifications (MIL-PRF-87269A) standardize structures for IETM content and give logical conditions for information rendering.

To create an IETM, authors and developers must consider a philosophy different from that used in to create page-based documents; they must implement behavioral aspects into the document. Unfortunately, documenting behavioral aspects into the specification itself, led to very complex specifications that few people actually tried to successfully implement such standards into actual functional IETMs.

On the other hand, S1000D (mentioned above) is a light and easy to understand specification. It contains minimal information about the behavioral aspects of the IETM, and focuses on structuring the core database information part. In doing so S1000D became popular among the IETM community in the sense that, the development time it takes to develop an IETM structured in accordance with S1000D is relatively low, when compared with the DoD specification (MIL-PRF-87269A). However, in omitting behavioral information, S1000D risked to take the interactivity factor out of the IETM, hence as mentioned before, the information being presented to the user tended to be overwhelming.

Interactivity is of primary concern to the IETM community. It is what truly makes IETMs a viable alternative to the ordinary paper manuals. Without Interactivity IETMs are simply reduced to electronically indexed versions of there paper counterparts.

One of the more compelling issues in IETMs is the issue of fault diagnosis. That is the diagnosis of a fault that could occur in the system the IETM documents. The users of IETMs often use such systems to see what could be the possible cause of a certain malfunction that occurred in a system. On the other hand, the IETM usually contains a sequence of steps a user must follow to ultimately reach the root cause of that particular fault. The IETM community has struggled to incorporate the concept of interactivity in that process. This means that we desire steps that a user follows to determine the cause of a fault be adapted to his level of expertise, and to his input, hence interactively displayed according to these factors. Up to this point, there have been two main approaches to achieve this:

1. *Hard coding such behavior in the specification itself*. The main problem with this approach is that, as mentioned before, it can lead to very complex specifications and very long authoring times.
2. *Leave the application of such behavior to the implementers of manual*. The main problem with this approach is that it could lead to inconsistent display of data. During the authoring of the IETM, the sequence of steps in the manual is validated by experts through actually performing the procedures documented on real systems and seeing if they actually reach the desired result. In leaving the reordering of such steps to the implementers of the manual we risk of displaying a sequence of steps that has not been validated and hence following it might not yield the desired result.

AECMA standard S1000D has opted to use the second approach. There are minimal guidelines in the standard itself that dictate how such behavior should be implemented.

Hence, as we can see that purely following one approach is not sufficient to achieve an optimal result, which is the least possible authoring time with the highest level of Interactivity.

Another issue of interactivity in IETMs is to guide the user through the information present in the IETM, to reach the desired goal. The size of data in an IETM is so large that it can run to be overwhelming to the user, if it were to be displayed at once. Thus it is of utmost importance that an IETM hide information not relevant to the user and only display information of interest, hence guiding the user through the information space of the IETM.

Thus the system must "adapt" itself to the user, pruning off information not relevant to the task the user is trying to accomplish. In doing so, we achieve our goal of reducing the complexity found in today's paper manuals.

1.3 Problem Statement

We identify the following problems:

1. *General Interactivity problems*:

- There is no standard technique for modeling the users of the IETM, i.e. there is no standard way to track the user of the system and document his behavior.
- There is no standard methodology for controlling interactivity in an IETM and hence providing mechanisms for adapting IETMs to users.
- There are no standard techniques for reorganizing fault diagnosis steps to best suit a specific user.

2. *IETM Specification S1000D problems:*

- *Ambiguity*: The SGML-centered specifications in S1000D are precise in their descriptions for the data element contents, but ambiguous with respect to the semantics for handling interactivity of the system.

- *Obscurity*: The specification does not realize a technical architecture that is easily recognized and implemented using common practice techniques.
- *Lack of presentation constructs:* The specification itself lacks the constructs that dictate how the information is to be presented to the user
- *Over Simplicity*: As odd as it sounds, S1000D is too basic to implement complex IETM systems.

1.4 Research Objectives

Our main research objectives are to:

1. Develop a methodology to deal with the issues of interactivity discussed above.
2. Develop a methodology for adaptive fault diagnosis in IETMs.
3. Suggest additions/modifications to AECMA S1000D to conceal its inherent problems of ambiguity, obscurity, lack of presentation constructs and of being over simplistic.

To solve the above problems our methodology should:

- Be able to document and track the user
- Be able to store system history, to be able to use such experience to better predict user interests and behavior.
- Be able to adapt the IETM at hand to the user using it as mentioned above.
- Be generic, i.e. with relative ease, it should be applicable to any IETM specification and not just the specification in our case study (S1000D).

1.5 Technical Approach

1. Develop an IETM Object Model [5]. As with any software project undertaking we must begin with a comprehensive modeling effort. To better understand the IETM

specification at hand, we will develop an IETM object model, using the UML (Unified Modeling Language) [7]. It will provide us with a graphical representation of the SGML specification, hence helping us to better visualize the constructs that make up the IETM.

2. Develop a framework that models the user. We shall present a framework for modeling the users of the IETM, providing user profiles that can be used to adapt the IETM.

3. Develop a methodology that adapts fault diagnostic steps to the user. We shall develop a complete methodology and framework that will take the user profiles, as well as other collected information and adapt the IETM fault procedures according to who is performing it.

4. Develop a Web Services enabled proof of concept system that implements our proposed methodology. Since the IETM itself will be structured using XML, we will implement a system that uses web services technology.

Part I: Preliminaries

In this section, we will present some background information that we deem necessary for the reader to have an idea about, before going on to explaining our approach in solving the problems outlined in chapter I.

CHAPTER 2: BACKGROUND

2.1 Interactive Electronic Technical Manual (IETM)

2.1.1 Overview

An Interactive Electronic Technical Manual (IETM) [1], is a package of information and documentation required for diagnosis and maintenance of complex weapon systems and both military and commercial equipment. An IETM possesses the following characteristics:

- *Electronic:* To enhance comprehension, an IETM contains information that is designed and formatted specifically for screen presentation.
- *Interactive:* The IETM display system operates interactively as a result of user requests and the associated input information.
- *Navigable:* The information in the manual must be linked together. The user must be able to reach an item inside the IETM from a multitude of different places. Information elements inside the IETM can contain references to other elements inside or even outside the IETM itself (for example links to information in other IETMs).
- *Portable:* The IETM must be portable, i.e. it must be accessible from as many systems as possible and easily transferable between systems.

IETMs are authored in a way that allows a user to retrieve and comprehend the required information faster and easier than is possible with their paper counterparts. IETMs mainly incorporate data that represent the knowledge of experts in the area. Information inside the IETM is of usually four main types:

1. *Descriptive:* Usually in this kind, you will find information describing certain aspects of the system, certain features, characteristics, etc.

2. *Procedural:* Information about procedures that could be performed on the system, like maintenance procedures, check ups, etc.

3. Fault Information: Information about faults that could occur in the system. How to diagnose them, how to find the root cause of the failure, and ultimately how to fix the fault.

4. Parts Information: Information about parts that make up the system, in a mechanical system these would describe the components and parts that work together to make the whole system.

Other than the data part of the IETM, some specifications contain information on how the IETM should be displayed to the user, although this is not the case with AECMA S1000D.

2.1.2 IETM Specifications

The US department of defense issued two main specifications relating to IETMs:

- MIL-PRF-87268A: Defines how IETM look and feel.
- MIL-PRF-87269A SGML Based [6]: Defines the IETM Data Base, i.e. the actual data that is found inside the manual.

The European Association of Aerospace Industries (AECMA) [3] has an IETM Specification called S1000D [4]. S1000D has the characteristic of being simple and light weight; hence it has gained popularity in the past years. We will explain S1000D thoroughly in subsequent sections and chapters.

2.1.3 Interactive Electronic Technical Manual Classes

IETMs are classified into six different classes, according to their complexity:

Class 0: Non-Electronically-Indexed Page Images

Systems of digitized images that are intended for electronic archival filing or paper printing. Class 0 IETMs are basically scanned images of their paper counterparts. They allow pages to be viewed on an electronic display but have no detailed index for navigation through the document. This system is not an IETM.

Class 1: Electronically Indexed Page Images

Systems of digitized page images intended for full-page display and use allowing navigation by means of an automated intelligent index to the page images. These systems are used in libraries or a reference setting for reading and research use.

Class 2: Electronically Scrolling Documents

Systems for Interactive display of ASCII encoded documents using an intelligent index and hypertext tags inserted into a tagged document file. In general, the document is the result of a simple conversion from a page oriented document but with little reauthoring with the exception of adding hypertext tags (links). These allow a user to navigate through the document, but have a very limited, if any, author inserted navigation aids or a content driven functions.

Class 3: Linearly Structured IETMs

Interactive display of technical information which is SGML tagged using an IETM specification, including, as much as possible hypertext links to connect the IETM to each other for easy navigation. It is linearly structured as a SGML document file, and not as a hierarchal structured data base.

Navigation is based on author developed constructs employing prompted dialog boxes and content driven logical navigation functions.

Class 4: Hierarchically Structured IETMs

Interactive electronic display of technical information specifically authored into and maintained in a relational or even an object oriented hierarchal database. These data are subsequently extracted and formatted for interactive presentation in accordance with

any IETM specification. This class, basically, provides the same functionality of class 3 IETMs.

Class 5: Integrated Database IETIS

Integrated Electronic Technical Information Systems (IETIS). Interactive presentation of class 4 IETMs integrated in with other processes including expert system rules for the display of information and other user-applications such as diagnostics or computer-managed training.

Each of these has benefits over the current paper technical manual systems and the degree of benefit increases with each higher class.

Realistic IETMs that are considered an alternative to technical manuals, span classes 3 through 5, they all show almost identical features at the user-presentation level. Systems of the Class 5 type exist but, in general, are proprietary in implementation and, typically include state of the art technology. They are designed to be a natural extension of Class 4 database IETMs.

In this work, we are mainly addressing the problems that tend to appear in Classes 3, 4 and 5.

2.2 SGML (Standard Generalized Markup Language)

SGML [6] is a standard for how to specify a document markup language or tag set. Such a specification is itself a document type definition (DTD). A DTD is basically a set of rules that govern how a specific instance of the SGML should be structured. SGML is not in itself a document language, but a description on how to use one; it is basically a Meta language. For example, HTML (Hyper Text Markup Language) is a DTD applied to SGML that defines all the HTML tags.

SGML was created to represent information in a neutral, self describing format. These properties of an SGML document make it ideal for insuring that information content in a document remains conformant to a specified set of user defined rules, hence making it easily accessible and organizable.

AECMA Specification S1000D is written in tags defined in a SGML.

2.3 Modeling and the Unified Modeling Language (UML)

Modeling is the designing of software applications before coding. Modeling is an Essential Part of large software projects, and helpful to medium and even small projects as well. A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper. Using a model, those responsible for a software development project's success can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code renders changes difficult and expensive to make. Surveys show that large software projects have a huge probability of failure - in fact, it's more likely that a large software application will fail to meet all of its requirements on time and on budget than that it will succeed. If you're running one of these projects, you need to do all you can to increase the odds for success, and modeling is the only way to visualize your design and check it against requirements before your crew starts to code.

The Unified Modeling Language (UML) [7, 11] helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements. (You can use UML for business modeling and modeling of other non-software systems too.) Using any one of the large number of UML-based tools on the market, you can analyze your future application's requirements and design a solution that meets them, representing the results using UML's twelve standard diagram types.

You can model just about any type of application, running on any type and combination of hardware, operating system, programming language, and network, in UML.

UML defines twelve types of diagrams, divided into three categories: Four diagram types represent static application structure; five represent different aspects of dynamic behavior; and three represent ways you can organize and manage your application modules.

Structural Diagrams include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.

Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and State chart Diagram.

Model Management Diagrams include Packages, Subsystems, and Models.

We will use the UML in our comprehensive modeling effort, explained in later chapters.

2.4 Internet and Web Technologies

2.4.1 Extensible Markup Language

XML [8] is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

What's a Document?

The number of applications currently being developed that are based on, or make use of, XML documents is truly amazing (particularly when you consider that XML is a relatively new technology)! For our purposes, the word "document" refers not only to traditional documents, like this one, but also to the myriad of other XML "data formats". These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and a thousand other kinds of structured information. In essence an IETM is a type of document as explained above.

So XML is Just like HTML?

No. In HTML, both the tag semantics and the tag set are fixed. An <h1> is always a first level heading and the tag <ati.product.code> is meaningless. Where as in XML, we can explicitly define the tag <ati.product.code>. The W3C [9], in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation (style sheets) to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount. And for people who want to disseminate information widely, features supported by only the latest releases of Netscape and Internet Explorer are not useful.

XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by style sheets.

The XML Document Object Model (DOM) is a programming interface for XML documents. It defines the way an XML document can be accessed and manipulated.

With the XML DOM, a programmer can create an XML document, navigate its structure, and add, modify, or delete its elements

2.4.2 Web Services

The basic idea behind Web services [10] is to adapt the loosely coupled Web programming model for use in applications that are not browser-based. The goal is to provide a platform for building distributed applications using software running on different operating systems and devices, written using different programming languages and tools from multiple vendors, all potentially developed and deployed independently. Web Services are just software components deployed on a web server.

As we said, Web Services are really just applications, so fundamentally they do what your applications do today. However, the way they do things is different. Consider three things that Web Services are especially good at:

- **Integration:** In most large organizations, the data and logic of one application are basically useless to other applications. When an application and its data are isolated from other applications, we often say that they are in "silos." In some of the most technology-savvy companies today it is not unusual to find a billing application that cannot ask a shipping application whether a delivery has been made. Application integration is one of today's most important business problems – one that can typically cost millions of dollars to resolve. In contrast, Web Services are better at sharing data and functions. The result is that the "silos" come down, and previously isolated systems can talk to each other, presenting enormous opportunities for improved business performance.
- **Access:** Web Services are especially good at providing access through different interfaces. A Web Service can have a dedicated client application, but it can also be readily accessed through browsers, wireless devices, voice-activated interfaces, and so on. Best of all, adding new access methods is much simpler than with a traditional application. For this reason, Web Services are especially useful when an application requires multiple access methods. For example, if an inventory

application can be accessed and updated through a Web portal, a wireless device, or even a customer's Web page, the business can accomplish more with its inventory system and data.

- **Flexibility***:* One of the most important innovations in Web Services is "machine-to-machine communications." This means that a Web Service can ask another Web Service to do something, and that Web Service can ask another Web Service to do something, and so on. In fact, many Web Services are really just aggregations of other Web Services. This can lead to some very complex situations, but it's fundamental to the promise of Web Services: in the future, all your information technology, and that of your business partners, can be called upon to respond to new customer needs and new market opportunities.

Web services build on the loose coupling of the traditional Web programming model, and extend it for use in other kinds of applications. There are three major differences between Web services and traditional Web applications: Web services use SOAP [12] (Simple Object Access Protocol, discussed later) messages instead of MIME (Multipurpose Internet Mail Extensions*)* messages [41], Web services are not HTTP-specific, and Web services provide metadata describing the messages they produce and consume. Let's consider each of these differences.

First, Web services communicate using SOAP messages. SOAP formalizes the use of XML as a way to pass data from one process to another. SOAP defines a framing model for protocol versioning and extensibility, a way to convey error information and a way to send messages over HTTP. The body of a SOAP message contains whatever XML an application wants to send.

The second major difference between Web services and traditional Web applications is that Web services are not transport protocol specific. While the SOAP specification only defines how to send SOAP messages over HTTP - and that's what the vast majority of today's Web services do - other transport protocols can also be used.

SOAP messages can be sent using SMTP, raw TCP, an instant messaging protocol like Jabber, or any other protocol you like. While most SOAP messages will be sent over HTTP for the foreseeable future, the ability to use other protocols is very important. HTTP was not designed to support long-running requests or sending event notifications to clients. These problems are best solved using other protocols, and standardized support for this will come over time.

Third, Web services are self-describing; they provide metadata describing the messages they produce and consume, the message exchange patterns they use to expose behaviors, the physical transport protocols they use, and logical addressing information required to invoke them. A Web service's message formats are defined using XML Schema (XSD). XML Schema is flexible enough to describe a wide range of message structures, including open content models with fine-grained control over extensibility, which is critical for services to be loosely coupled in terms of the data they send and receive. A Web service's behaviors are described using the Web Service Description Language (WSDL), which map message exchanges to operations grouped into interfaces and describe how those operations can be invoked using particular transport protocol bindings. You use these descriptions to write software that communicates with a service, either directly or indirectly via some code.

2.4.3 Simple Object Access Protocol (SOAP)

SOAP [12] provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC (Remote Procedure Call).

SOAP consists of three parts:

- The SOAP envelope construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.
- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

Although these parts are described together as part of SOAP, they are functionally orthogonal. In particular, the envelope and the encoding rules are defined in different namespaces in order to promote simplicity through modularity.

In addition to the SOAP envelope, the SOAP encoding rules and the SOAP RPC conventions, this specification defines two protocol bindings that describe how a SOAP message can be carried in HTTP messages either with or without the HTTP Extension Framework

A major design goal for SOAP is simplicity and extensibility. This means that there are several features from traditional messaging systems and distributed object systems that are not part of the core SOAP specification. Such features include

- Distributed garbage collection
- Batching of messages
- Objects-by-reference (which requires distributed garbage collection)
- Activation (which requires objects-by-reference)

2.4.4 .Net Framework

To quote from the Microsoft Web Site [13]: "Microsoft® .NET is a set of Microsoft software technologies for connecting information, people, systems, and devices. It enables a high level of software integration through the use of XML Web services—

small, discrete, building-block applications that connect to each other as well as to other, larger applications over the Internet".

Basically, Microsoft's .Net framework is an environment for developing applications. It is a set of libraries that run under Microsoft's Windows Operating System. Built on top of this framework is a set of programming languages that all use the libraries resources. These Languages include C#, J# (Microsoft's version of JAVA"), Visual Basic. Net and C++ .Net. What distinguishes this suite of programming languages is the fact that they all compile to the same intermediate language called IL. Then this language is handled by the same runtime environment called the Common Language Runtime (CLR). Visual Studio. Net is the main integrated development environment (IDE) that handles all the .Net languages. It is software that has a GUI interface primarily built by Microsoft to develop application written in any .Net supported language.

2.5 Paradigms of Expert and Intelligent Systems

2.5.1 Rule-Based Systems

A rule is an expression of the form:

If *A* **then** *B*

where *A* is an assertion and *B* can be either an action or another assertion. For instance, the following three rules could be part of a larger set of rules for troubleshooting water pumps:

1) **If** pump failure **then** the pressure is low

2) **If** pump failure **then** check oil level

3) **If** power failure **then** pump failure

A *rule-based system* consists of a library of such rules. These rules reflect essential relationships within the domain, or rather: they reflect ways to reason about the domain.

When specific information about the domain becomes available, the rules are used to draw conclusions and to point out appropriate actions. This is called *inference*. The inference takes place as a kind of chain reaction. In the above example, if you are told that there is a power failure, Rule 3 will state that there is a pump failure and Rule 1 will then tell us that the pressure is low. Rule 2 will also give a (useless) recommendation to check the oil level.

Rules can also be used in the opposite direction. Suppose you are told that the pressure is low; then Rule 1 states that it can be due to a pump failure, while Rule 3 states that a pump failure can be caused by a power failure. You should also be able to use Rule 2 to recommend checking the oil level, but it is very difficult to control such a mixture of inference back an forth in the same session.

2.5.2 Uncertainty

Often the connections reflected by the rules are not absolutely certain, and also the gathered information is often subject to uncertainty. In such cases, a *certainty measure* is added to the premises as well as to the conclusions in the rules of the system. Now, a rule gives a function that describes how much a change in the certainty of the premise will change the certainty of the conclusion. In its simplest form, this looks like:

If *A* (with certainty *x*) **then** *B* (with certainty *f(x)*)

There are many schemes for treating uncertainty in rule-based systems. The most common are *fuzzy logic, certainty factors,* and (adapted versions of) *Dempster-Shafer belief functions*. Common to all of these schemes is that uncertainty is treated locally. That is, the treatment is connected directly to the incoming rules and the uncertainty of their elements. Imagine, for example, that in addition to Rule 4 we have the rule

If *C* (with certainty *x*) **then** *B* (with certainty *g(x)*)

If we now get the information that *A* holds with certainty *a* and *C* holds with certainty *c*, what is then the certainty of *B*?

There are different algebras for such a combination of uncertainties, depending on the scheme. Common to all these algebras is that in many cases they come to incorrect conclusions. This is because the combination of uncertainty is not a local phenomenon, but it is strongly dependent on the entire situation (in principle a global matter).

2.5.3 Neural Networks

A neural network consists of several layers of *nodes*: At the top there is a layer of *input nodes*, at the bottom a layer of *output nodes,* and in between these normally 1 or 2 *hidden layers*. Except for the output nodes, all nodes in a layer are in principle connected to all nodes in the layer immediately below. A node along with its in-going edges is called a *perceptron*.

A neural network performs *pattern recognition*. You could for instance imagine a neural network that reads handwritten letters. By automatic tracking, a handwritten letter can be transformed into a set of findings on curves (not a job for the network). The network will have an input node for every possible kind of finding and an output node for each letter in the alphabet. When a set of findings is fed into the network, the system will match the pattern of findings with equivalent patterns of the different letters.

Technically, the input nodes are given a value (0 or 1). This value is transmitted to the nodes in the next layer. Each of these nodes perform a weighted sum of the incoming values, and if this sum is greater than a certain threshold, the node fires downward with the value 1. The values of the output nodes determine the letter.

So, apart from the architecture of the network (the number of layers and the number of nodes in each layer), the *weights* and the *thresholds* determine the behavior of the

network. Weights and thresholds are set in order for the network to perform as well as possible. This is achieved by *training*: You have a large number of examples where both input values and output values are known. These are then fed into the training algorithm of the network. This algorithm determines weights and thresholds in such a way that the distance between the set of outputs from the network and the desired sets of outputs from the examples gets as small as possible.

There is nothing preventing the use of neural networks for domains requiring the handling of uncertainty. If relations are uncertain (for example in medical diagnosis), a neural network with the proper training will be able to give the most probable diagnosis given a set of symptoms. However, you will not be able to read the uncertainty of the conclusion from the network, you will not be able to get the next-most probable diagnosis and - probably the most severe set-back - you will not know under which assumptions about the domain the suggested diagnosis is the most probable.

2.5.4 Bayesian Networks

Bayesian networks [14] are also called *Bayes nets, causal probabilistic networks* (CPNs), *Bayesian belief networks* (BNs), or *belief networks*.

A Bayesian network consists of a set of *nodes* and a set of directed *edges* between these nodes. Edges reflect cause-effect relations within the domain. These effects are normally not completely deterministic (e.g. disease -> symptom). The *strength* of an effect is modeled as a probability:

6) **If** tonsillitis **then** P (temp>37.9) = 0.75

7) **If** whooping cough **then** P (temp>37.9) = 0.65

One could be led to read these statements as rules. They shouldn't. So, a different notation is used:

P(temp>37.9 | whooping cough) = 0.65

If 6) and 7) are read as '**If** otherwise healthy and...**then**...', there also needs to be a specification of how the two causes combine. That is, we need the probability of having a fever if both symptoms are present and if the patient is completely healthy. All in all you have to specify the *conditional* probabilities:

P (temp>37.9 | whooping cough, tonsillitis),

Where 'whooping cough' and 'tonsillitis' each can take the states 'yes' and 'no'. So, you must for any node specify the strength of all combinations of states for the possible causes.

Fundamentally, Bayesian networks are used to update probabilities whenever information becomes available. The mathematical basis for this is Bayes' theorem:

P(A | B) P(B) = P(B | A) P(A)

Contrary to the methods of rule-based systems, the updating method of Bayesian networks uses a global perspective, and if model and information are correct, it can be proved that the method computes the updated probabilities correctly (correctly regarding the axioms of the classical probability theory).

Any node in the network can receive information as the method doesn't distinguish between inference in or opposite to the direction of the edges. Also, simultaneous input of information into several nodes will not affect the updating algorithm.

An essential difference between rule-based systems and systems based on Bayesian networks is that in rule based systems you try to model the expert's way of reasoning (hence the name *expert systems)*, while with Bayesian networks you try to model dependences in the domain itself. Systems of the latter type are often called *decision support systems* or *normative expert systems*.

2.5.5 Comparing Neural Networks and Bayesian Networks

The fundamental difference between the two types of networks is that a perceptron in the hidden layers does not in itself have an interpretation in the domain of the system,

whereas all the nodes of a Bayesian network represent concepts that are well defined with respect to the domain.

The meaning of a node and its probability table can be subject to discussion, regardless of their function in the network. But it does not make any sense to discuss the meaning of the nodes and the weights in a neural network. Perceptrons in the hidden layers only have a meaning in the context of the functionality of the network.

This means that the construction of a Bayesian network requires detailed knowledge of the domain in question. If such knowledge can only be obtained through a series of examples (i.e., a data base of cases), neural networks seem to be an easier approach. This might be true in cases such as the reading of handwritten letters, face recognition, and other areas where the activity is a 'craftsman like' skill based solely on experience.

It is often criticized that in order to construct a Bayesian network you have to 'know' too many probabilities. However, there is not a considerable difference between this number and the number of weights and thresholds that have to be 'known' in order to build a neural network, and these can only be learnt by training. It is an enormous weakness of neural networks that you are unable to utilize the knowledge you might have in advance.

Probabilities, on the other hand, can be assessed using a combination of theoretical insight, empiric studies independent of the constructed system, training, and various more or less subjective estimates.

Finally, it should be mentioned that in the construction of a neural network the route of inference is fixed. It is decided in advance, about which relations information is gathered, and which relations the system is expected to compute. Bayesian networks are much more flexible in that respect.

CHAPTER 3: RELATED WORK

3.1 Introduction

There are many research projects that deal with adaptable systems, namely in the area of adaptive hypermedia systems [22]. These systems have mainly focused on adapting websites that provide customer support. Such systems are presented in [35], [36] and [37]. Such systems provide adaptation techniques for adaptive web sites to specific users to provide customer support.

Another area of interest has in adaptive systems is user modeling. This represents the activity of tracking the user as he navigates through the system and documenting his behavior. Such techniques are presented in [38], [39], and [40]. These mainly present user modeling techniques for adapting web based systems, while [40] presents an empirical study on the usage of dynamic Bayesian Networks for user modeling.

Moreover, of utmost relevance to our work, is in the area of adaptivity related to fault diagniosis or to IETMs in general. Adapts [15] , is an electronic support system that provides adaptive diagnostics. Lumiere [20] is a Microsoft research project in which the Microsoft office assistant (clippie) was the direct result of. SACSO [17] is also a research project sponsored by Hewlett Packard and mainly deals with fault diagnosis in printing systems. All of these systems are discussed in the following sections.

3.2 Adaptive Diagnostic and Personalized Technical Support (ADAPTS)

ADAPTS [15] is an electronic performance support system that integrates an adaptive diagnostics engine with adaptive access to supporting information. Integrated performance support systems bring together an expert system-like problem solving engine and an on-line information system. ADAPTS provides comprehensive adaptive support on several stages of troubleshooting from identifying the source of troubles to determining the course of actions to guiding the user through the troubleshooting process to assembling the individualized set of supporting materials.

The ADAPTS system was initially developed as a proof-of-concept research project using operational technical manual data from a Navy H-60 helicopter program. During this initial phase, the focus was on developing the adaptive user interface, integrating the Condition Based Maintenance (CBM) software, and experimenting with the response of the system to variations in the user model. The first version of the system was implemented in 1999. More recently, a follow-up contract was granted to review the usability of the system in both an aiding (performance-oriented maintenance assistance) and training contexts.

ADAPTS uses a collapsible checklist of steps to guide the technician through a troubleshooting procedure. ADAPTS determines how to present this checklist based on a dynamic assessment of the user's expertise with that procedure. For example, ADAPTS collapses a subtask outline if the technician is experienced with the subtask. Inexperienced technicians automatically receive an expanded outline of subtasks that reveals details. Experienced technicians may expand the outline if they choose, and are given greater flexibility to navigate within the checklist. Inexperienced technicians are given more assistance in step-by-step navigation. As a technician completes a step within the checklist, color-coding and icons identify completed, current, and remaining steps.

3.3 Microsoft's Lumiere

Uncertainty is ubiquitous in attempts to recognize an agent's goals from observations of behavior. The Lumiere project [20] at Microsoft Research is focused on leveraging methods for reasoning under uncertainty about the goals of software users. At the heart of Lumiere research and prototypes are Bayesian user models that capture the uncertain relationships among the goals and needs of a user and observations about program state, sequences of actions over time, and words in a user's query. Motivating problems include the computation and use of probability distributions over a user's goals for providing appropriate assistance or automated services, for

dynamically tailoring language models in speech recognition, and for appropriately guiding the allocation of computational resources in an operating system.

Lumiere has been applied in the office assistant application in Microsoft's Office products. It basically builds a Bayesian user model that models the user's actions and suggests "help topics" that the user is interested in. This work is mainly concerned with trying to relate topics of help information to actions that indicate user's interests in those topics.

3.4 SACSO

SACSO (Systems for Automated Customer Support Operations) [17] is a collaboration between the research unit of Decision Support Systems at Aalborg university and customer support R&D at Hewlett Packard Company. SACSO is a Bayesian Network tool for automated diagnosis of printing systems. It has been specifically designed to model printers to guide users trying to troubleshoot a printer.

SACSO models the printing system in a Bayesian Belief Network (refer to chapter 2).

Part II: Approach

In this part, we represent our methodology for solving the problems outlined in chapter one. We will start by explaining our modeling approach to reach the goal of establishing a framework for interactivity and adaptivity in IETMs. We will then present our approach for adaptive fault diagnosis.

CHAPTER 4: SYSTEM MODELING

4.1 IETM Specification Overview

AECMA S1000D [4] is an International Specification for Technical Publications utilizing a Common Source Data Base (CSDB) and is used for the procurement and production of technical publications. Whilst the title restricts its use to technical documentation it has been demonstrated that the principles of the specification can easily be applied to non-technical documentation. The specification adopts and profiles ISO [20] and WWW standards. Information generated is in neutral format, which means it can be used on disparate IT systems. It is this feature together with the modular approach to data creation and storage that makes the specification so acceptable to the wider international community.

The specification is currently at Issue 1 Change 9. It incorporates a methodology for storing data in electronic form and provides the capability to output information both in electronic and, if required, paper format. The information types that can be handled are:

- Descriptive
- Fault Isolation
- Procedural
- Crew/operators
- Maintenance schedules
- Parts data

Data produced to AECMA S1000D is presented in a modular form (data modules). Data modules are defined as "self contained units of data" and are stored in a database called the Common Source Database (CSDB). Individual data modules are identified by a logical and specific numbering system, the Data Module Code (DMC),

which permits the use of a database to store and manage the complete information set.

Data modules have two sections: one containing the content, which is the data, required by the user e.g. a description or procedure, the other is the Identification and Status section, which contains all the metadata necessary to control and identify the data module and its configuration. Each item of information, therefore, carries all its own configuration data.

A project's complete technical publications information set is held on a Common Source Data Base (CSDB). The combination of data module code, information types and DM metadata allows a selection of subsets of information to be chosen by query or table of contents designed to meet a specific users needs.

4.1.1 Characteristics of S1000D

The specification has the following characteristics that make it appealing for developing IETMs:

- It is based on internationally agreed neutral standards.
- It reduces maintenance costs for technical information.
- It allows sub-sets of information to be generated to meet specific user needs.
- It allows transfer of information and electronic output between disparate IT systems.
- Many different output forms can be generated from the same base data set ensuring strong, efficient data configuration control at the user interface.
- The AECMA S1000D data module concept can be and has been applied to legacy data.
- It is non-proprietary.
- It allows neutral delivery and management of data.
- It uses the CALS philosophy of "create once use many times".
- It is in daily use over many national and international projects.
- The modular approach is ideal for web delivery.

- Responsive to emerging technology.

4.2 IETM Object Model

Modeling is a proven and well accepted software engineering technique. Good modeling is essential to assure architectural soundness and to ease of use between a software project's team. We need to build a model of the IETM complex system, because it will be a cumbersome task to try to understand it in its entirety. So as the complexity of the system increases, modeling techniques become more effective. Moreover, the resultant model enables us to explore multiple solutions effectively.

So we will develop an IETM object model [5] that will help us understand the specification in a more comprehensive way. The method we use to develop the object model is called "relaxed DTD mapping" (figure 4.1) as described in [21].

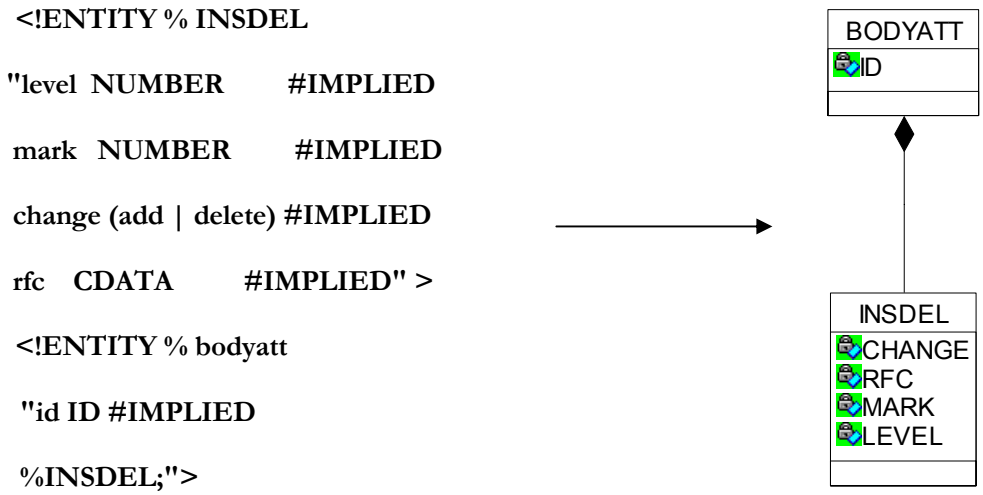


Figure 4.1: Relaxed DTD Mapping

4.2.1 IETM structure

As mentioned previously, the IETM specification is divided into smaller information units called data modules, grouped in a Common Source Data Base (CSDB).

A data module is composed of the following parts (figure 4.2):

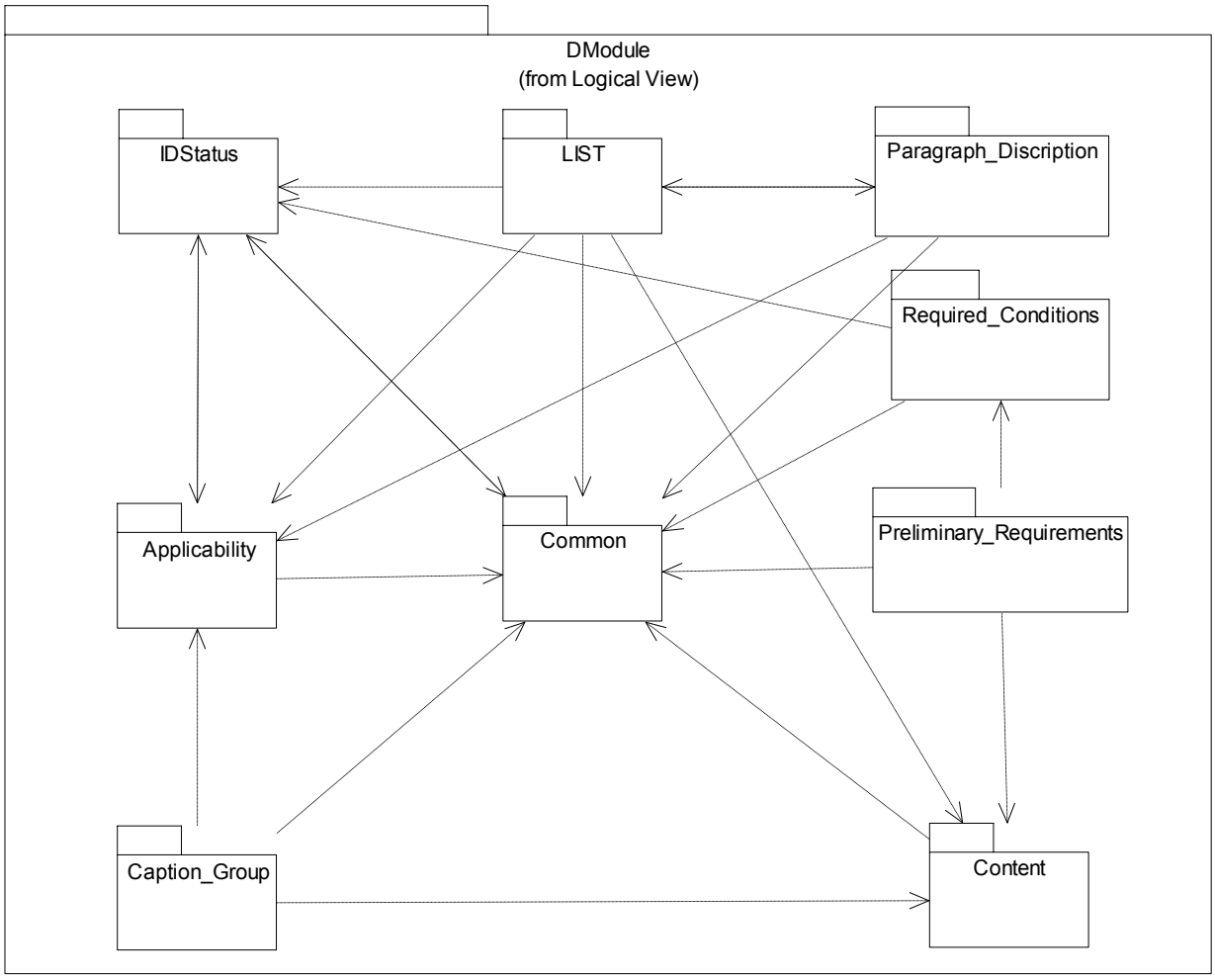


Figure 4.2: Top Level Package Structure of S1000D

1. Identification Part.
2. Content Part.
3. List Structure.
4. Paragraph Structure.

5. Required Conditions.
6. Preliminary Conditions.
7. Caption Group.
8. Applicability.
9. Common Elements

The `identification` part (See appendix A) provides a structure for giving data modules unique Ids, so that they can be addressed and referenced from inside the IETM they belong to, or even from an outside IETM. So a data module must have an ID that is unique across all existing IETM CSDBs.

The `Content` Part (figure 4.3) is the actual data in the data module. It is sub divided

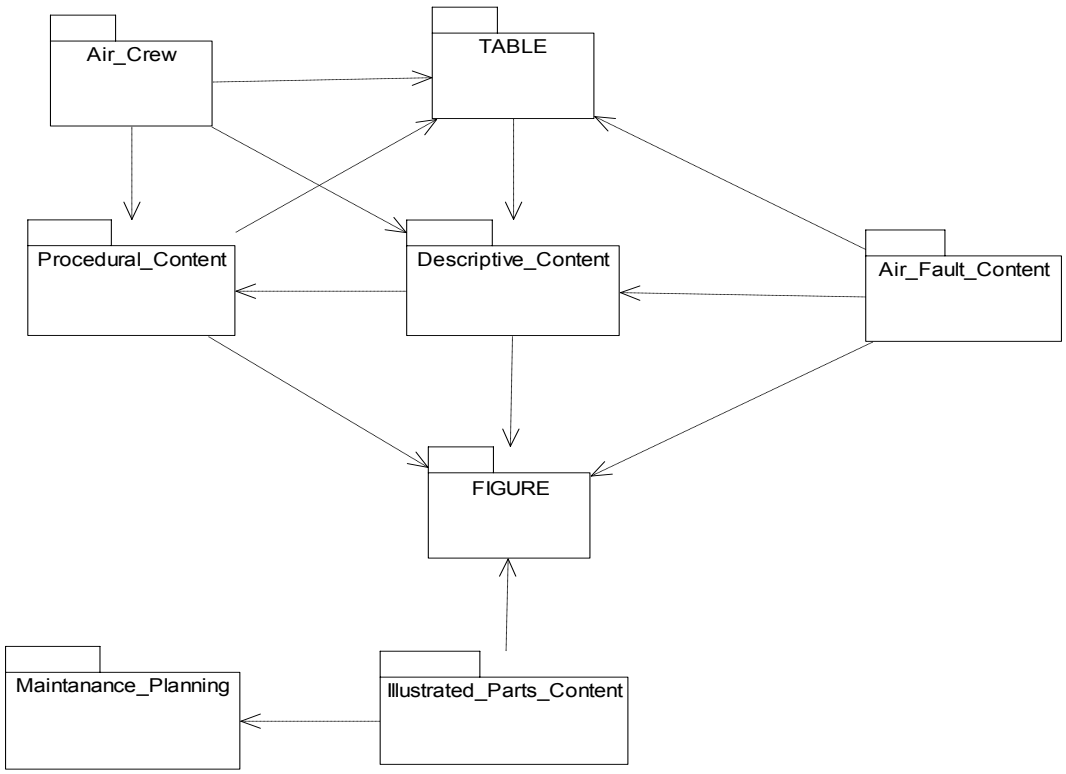


Figure 4.3 Content Package

into several packages that represent the different types of information present in the data modules. These are:

1. The `Air Fault` Package: This package basically describes any faults that might occur in an air vehicle. It describes the fault, how to diagnose it, and how to fix it. This package is subdivided into two packages (figure 4.4):
    a. `Air Fault Isolation` : This package contains information on how air faults should be described, and how procedures to isolate (detect) the fault should be documented
    b. `Air Fault Reporting`: How an observed fault in the system should be reported and documented.

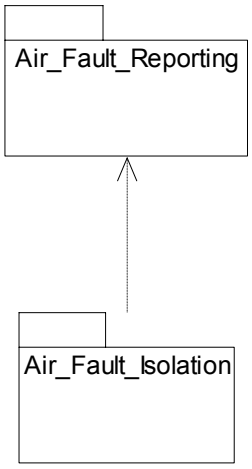


Figure 4.4: Air Fault Package

2. `Descriptive Content` Package: This Package dictates how descriptive information, i.e. information that describes certain aspects and components of the system, should be structured (see appendix A).
3. `Procedural Content` Package: This Package dictates how information of procedural nature, i.e. information that describe how to perform certain procedures on components of system, should be structured (see appendix A).

4. `Air Crew` Package: This Package dictates how information about the air crew that is using the IETM should be structured (see appendix A).
5. `Illustrated Parts Package`: This Package dictates how information about the parts that make up the air vehicle should be structured.
6. `Maintenance Planning` Package: This Package dictates how information regarding the planning of maintenance procedures that are performed on air vehicles should be structured.
7. `Table` Package: Describes how tables should be represented in the data module.
8. `Figures`: Describes how figures inside IETMs should be represented.

The `List` package (figure 4.2) describes how lists that describe ordered structures should be represented in the data module. (See appendix A).

The `Caption Group` describes how captions are represented in the data module. (See appendix A)

The `Preliminary Requirements` package defines all the requirements that must be met before a task can be accomplished.

The `Required Conditions` package defines all the conditions that must

The `Common` package contains elements and objects that are shared among all packages.

Last but not least, it is important to note that a single data module unit can contain only one `IDSTATUS` section and one `CONTENT` section. As a result, a data module, at any one time, can contain only one type of information in its `CONTENT` section (`descriptive, procedural, maintenance, fault` or `aircrew`).

4.2.2 IETM Usage

As depicted in figure 4.5, a user of the IETM can be of three kinds:

- Novice
- Expert
- Designers

Novice users are beginners who have little experience in using IETMs or little knowledge of the data that is found inside the IETM.

Expert users are professionals with using IETMs and have extensive hands on experience with the procedures documented inside the IETM

Designers are users that actually design the IETM. They are responsible for structuring and authoring the IETM. They might not be knowledgeable with the information actually documented inside the IETM, but they are experts in IETM technology and with authoring such systems.

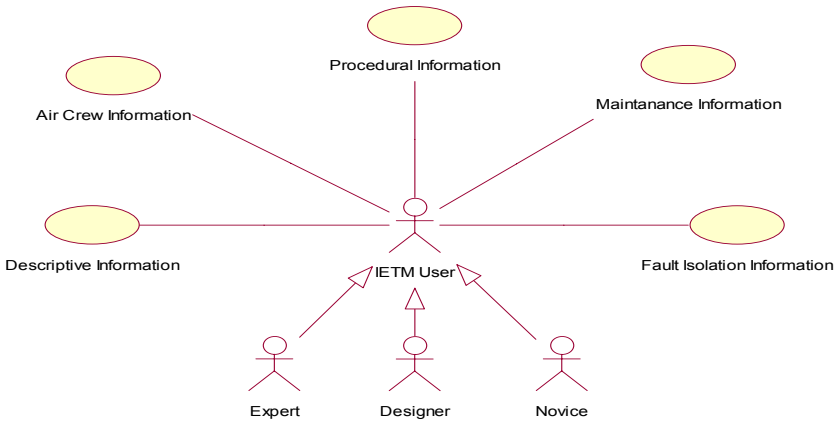


Figure 4.5: Usage of the IETM

Users of the IETM can request five kinds of information: Descriptive, procedural, Air crew, maintenance and fault information. When a user requests a specific data module, the whole data module with all its elements are returned (figure 4.6).

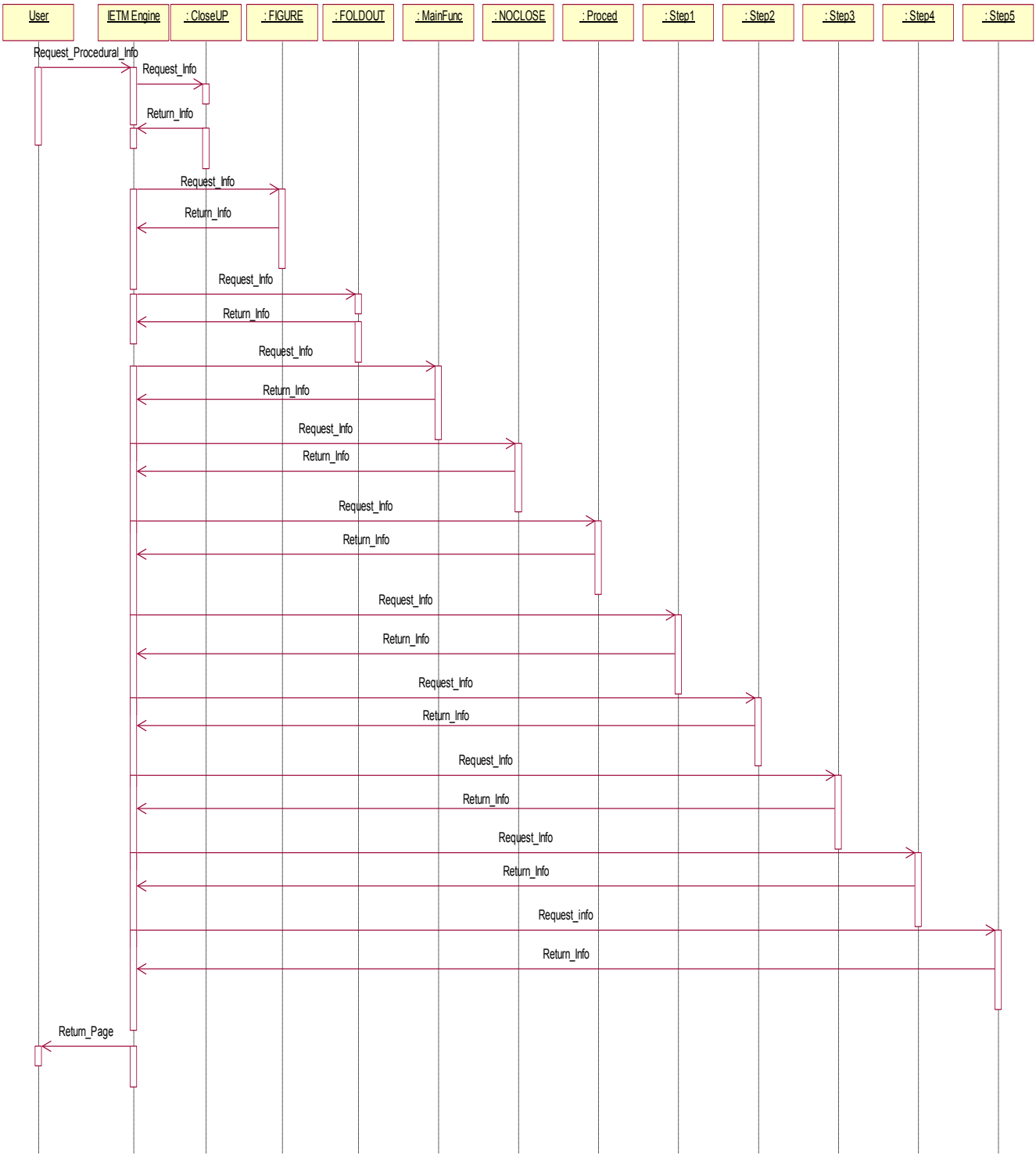


Figure 4.6: Requesting Procedural Information

In figure 4.6, we see that a user has requested a data module which contains procedural information. As we can see all the elements that make up procedural information are returned upon request.

The specification, as it stands now, reacts to the user as discussed above. It simply returns everything inside the data module, which as discussed in chapter one can be over whelming to a user. So we see that the degree of interactivity with the user is not sufficient to make the IETM truly responsive to the user's inputs and interests. To truly be interactive with the user we must first collect as many information about him as possible, and about all the users of the IETM, and try to "learn" from the interactions of others and the interactions of experts with the system to guide the current user of the system, who might be a novice. In the next section we shall describe a user model, which we will use in our methodology for establishing interactivity and adaptivity within IETMs.

4.3 User Modeling

4.3.1 Overview

User modeling [22] is that act of tracking and documenting user behavior with respect to the system. In order to develop a complete user model, or a framework for modeling the user, we must first consider what we are adapting to. That is what aspects of the user working with the system can be taken into account when providing adaptation. To which features, that can be different for different users (and may be different for the same user at different time), can the system adapt? Generally, there are many features related to the current context of the user work and to the user as an individual which can be taken into consideration by an adaptive system. These include the user's goals and the user's knowledge. We will discuss briefly what we mean by these considerations.

4.3.2 User Modeling Considerations

- *Knowledge*

User's knowledge of the subject represented in the IETM appears to be the most important feature of the user for existing adaptive systems. Almost all adaptive techniques rely on user's knowledge as a source of adaptation. User's knowledge is a variable for a particular user. This means that an adaptive system which relies on user's knowledge has to recognize the changes in the user's knowledge state and update the user model accordingly. User's knowledge of the subject is most often represented by an overlay model.

The idea of the overlay model is to represent an individual user's knowledge of the subject as an "overlay" of the domain model. For each domain model concept, an individual overlay model stores some value which is an estimation of the user knowledge level of this concept. This can be just a binary value (known–not known), a qualitative measure (good-average-poor), or a quantitative measure, such as a probability that the user knows the concept. An overlay model of user knowledge can be represented as a set of pairs "concept - value", one pair for each domain concept.

- *Goals and Interests*

User's goal or a user's task is a feature related with the context of a user's work in the system rather than with the user as an individual. Depending on the kind of system, it can be the goal of the work (in application systems), a search goal (in information retrieval systems), and a problem solving or learning goal (in educational systems). In all of these cases the goal is an answer to the question "Why is the user using the hypermedia system and what does the user actually want to achieve?" A User's goal is the most changeable user feature: almost always it changes from session to session and often can change several times within one session of work.

Next, we will present our proposed methodology for establishing a model for user interaction.

4.3.3 Object Oriented User Profiles

The main concept behind our approach is the use of the Unified Modeling Language (UML) to model the interactions of a user with the system. The advantage of this is that UML is a graphical notation, thus making it easy to comprehend to the designers of the IETM. This is essential, because the designers of the IETM need to understand how users interact with their system, in order to keep enhancing the design and structure of their system to make it easier for the users to navigate through the IETM. As mentioned in chapter 2, the UML has been used, ever since its invention by the Object Management Group (OMG) [23], to model artifacts of software systems. It models the logical structure of the software, through class diagrams, package diagrams and the like. It also models the behavior of the system, through the types of diagrams presented in chapter 2. Hence, we see that it constitutes a good decision to use the UML to model behavior of user with respect to the IETM.

As mentioned in the previous section, the most common model type is use today is the overlay model. The overlay model has the distinct disadvantage of not providing enough information about the actual sequence of navigation that the user went through while using the system. This is of crucial importance in the domain of IETMs, since, the designers of the IETM must design the system in such a way that minimizes as much as possible the number of items that the user must navigate through in order to reach his ultimate goal. This is impossible to observe if you are simply using an overlay model.

In this light, we will use the following UML diagrams to model the user's behavior:

- *Use case diagrams*

We will use "Use Case" diagrams to model user's interests in the system. Hence, as the user navigates through the system, he requests units of information that might be of interest to him; hence we will model these interests as use cases (figure 4.7).

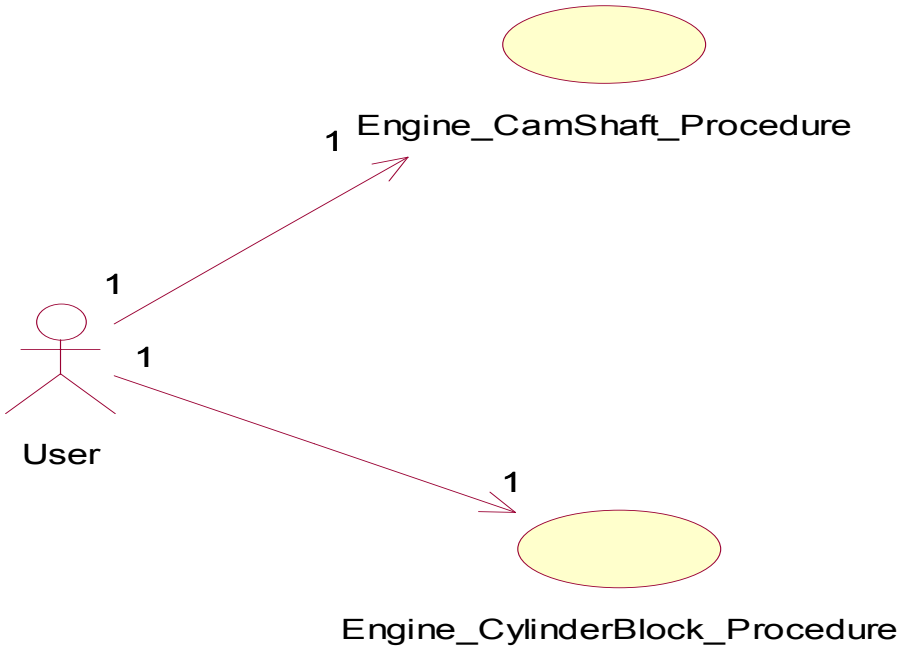


Figure 4.7: Use Case diagram documenting user interests

Thus, as we see in figure 4.7, the user has requested two procedures (the two use cases). These represent procedures that the user has requested, or displayed interest in, during his navigation inside the IETM.

• *Sequence Diagrams*

We will use sequence diagrams to model user's interactions with the system. As the user navigates through the system, it is pertinent, that we track the user and see how he navigates in the system, in order to use it to try to guide future users, who might be interested in the same information that this user is interested in, while they navigate through the system (refer to chapter 5).

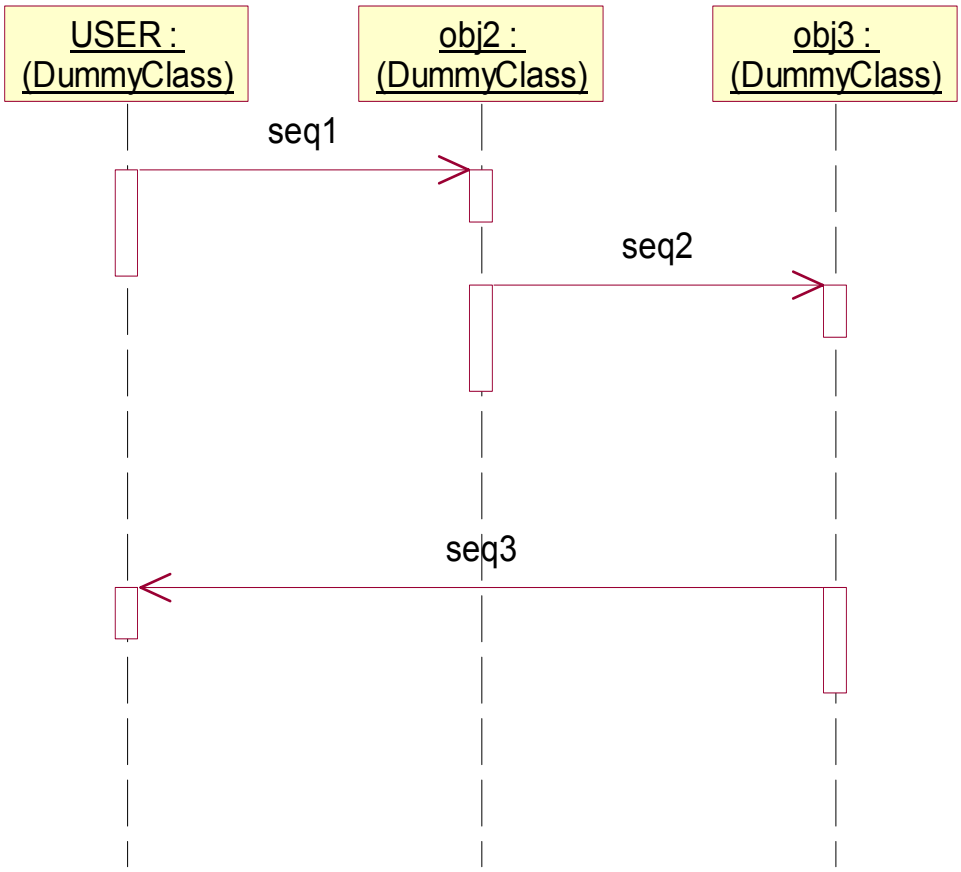


Figure 4.8: Sequence Diagram representing the user's interaction

In figure 4.8 we see that the user requested object 2, which in turn requested object 3, which is finally returned to the user. Hence, if a large number of users who requested "obj2", also requested "obj3", then it might be pertinent to guide future users in the same manner also, that is if they requested "obj2", the system might suggest that "obj3" might be of interest also.

Moreover, UML can be represented in XMI (Extensible Metadata Interchange) [24], which is a textual XML representation of UML. It provides a convenient way to interrogate UML models so that we can get all the data that is required from the users profiles depicted above.

Now that we have established a technique to document user's behavior, we shall move on to provide an adaptive framework for adapting fault diagnostic steps to a specific user.

CHAPTER 5: ADAPTIVE FAULT DIAGNOSIS

5.1 Overview

As stated previously, an IETM is massive repository of different kinds of information that are related to the description of the various aspects of the system that the manual is documenting. Among those types of data is fault diagnosis data. Fault diagnosis refers to the action of trying to find the cause of a specific fault in the system. A fault refers to any problem that can occur in the system. A fault is usually observed by the individual operating the system that is documented in the IETM. In order to continue with the normal operation on the system, the user must try to locate the source of the observed fault, and try, if possible to fix it.

To correctly reach the root cause of a fault the user usually has to go through a sequence of steps that must be performed or observations that must be observed. The main problem with today's IETMs is that the sequence that the user has to go through is fixed and hard coded into the manual. Current implementations of IETMs do not take into account the level of experience of the user. This is essential, because each step along the diagnostic sequence might vary in its complexity, and hence might be hard for some users to execute correctly. Hence the sequence must be able to adapt, i.e. the steps rearranged, in order to best fit the user actually performing the diagnostic procedure.

In this chapter we shall propose a methodology for adapting the sequence, as mentioned above. Our methodology will take into consideration the user's characteristics as well as other factors. In the next section we will thoroughly describe the proposed methodology.

5.2 Methodology

5.2.1 Adaptation Factors

In order to arrive at a methodology that correctly adapts a sequence of diagnostic steps to a specific user, the system must be able to adapt to several characteristics and factors. These factors represent data that must be collected and used in our methodology. These factors are:

- *User's experience*

The first and most important of the factors to consider for adapting the diagnostic process is user's experience. The user's experience refers to the expertise of the user or technician performing the diagnostic procedure. For example, users with low expertise prefer a sequence which is easier to perform, although it might take more time, while experienced users prefer to perform the procedure as fast as possible, although individual steps might be harder and more complex.

- *Step difficulty*

In order to adapt the diagnostic procedure to the user, the level of difficulty of a step must be taken into consideration. The difficulty of a step is proportional to the level of complexity of performing the step. Thus if a step requires more time, more expertise then it is more difficult to perform. Step difficulty is also relative to a user. Thus a step that is easy to perform, with regards to an experienced user, might be difficult to perform for a novice one. It is prudent to state that this measure is subjective, i.e. the level of difficulty of a step is provided to us by the authors that originally authored the IETM, based on subjective measurement provided by experts in the field.

5.2.2 Methodology Overview

Our methodology is primarily composed of three steps, the first two steps are mainly concerned with how we choose to represent our data, while the adaptation mechanism is represented in the final step.

These steps are:

1. *Modeling the diagnostic procedure*. As stated earlier, the diagnostic procedures found in IETMs are represented in SGML format. While this format is good for structuring data we shall represent diagnostic procedures in a form of a Bayesian Belief network as indicated in [14].

2. *Modeling the user*: We shall use the method outlined in the previous chapter to capture the user behavior in the system, and shall take all the information we need in our adaptation framework from the user profiles we established in the previous chapter.

3. *Adaptation methodology*. Develop a methodology to see which step in the diagnostic procedure to present next to the user. This is the core of our methodology. We shall develop a framework that takes the data modeled in steps (1) and (2) and apply techniques which will be described in detail in the following sections, to achieve a framework that will give us the next step to be performed in a diagnostic procedure with respect to a specific user.

5.2.3 Modeling diagnostic procedures overview

In order to better understand our technique, we will present a diagnostic procedure, and apply our methodology to it as we go.

The following diagnostic procedure, which we shall refer to as "**proc1**", which is partly based on the diagnostic procedure stated in [14], should do for our purpose:

"In the morning, my car will not start. I can hear the starter turn, but nothing happens. There may be several reasons for this problem. I can hear the starter roll, so there is nothing wrong with the battery of the car. Therefore, I might be out of gas, so I check the fuel meter, the fuel meter indicates that I have a full tank, so I rule that out. It

might be a serious problem like the fuel injection might be faulty or my spark plugs might need cleaning, or even replacing. It might be also that I may have a leak in the injection system. So I decide to see if the spark plugs are dirty, I removed the sparkplugs but find that they are clean. But then I remember that I have an alarm system that must be turned off, or the car won't start. I observe that the alarm light on the dash board is still on, so I turn off the alarm and start the car again, it starts and I am off to work!!"

The above is an example of reasoning that humans do daily. To have an automated system to do the same kind of reasoning, we need answers to questions such as: "What made me conclude that "out of fuel" or "battery is faulty" are the most probable causes for my problem" or "what me made me conclude that I must look at the fuel meter?". To be more precise, we need ways of representing the problem and ways of performing inference in this representation such that an automated system can simulate that kind of reasoning and perhaps do it faster and more efficient than a human. Thus, in the above case if I had deduced that the alarm might be on, it might have saved me the trouble of inspecting the state of the spark plugs.

Thus we can see that diagnostic procedures are procedures that a technician must follow to diagnose an observed fault. In a diagnostic procedure, steps are usually of three kinds (figure 5.1):

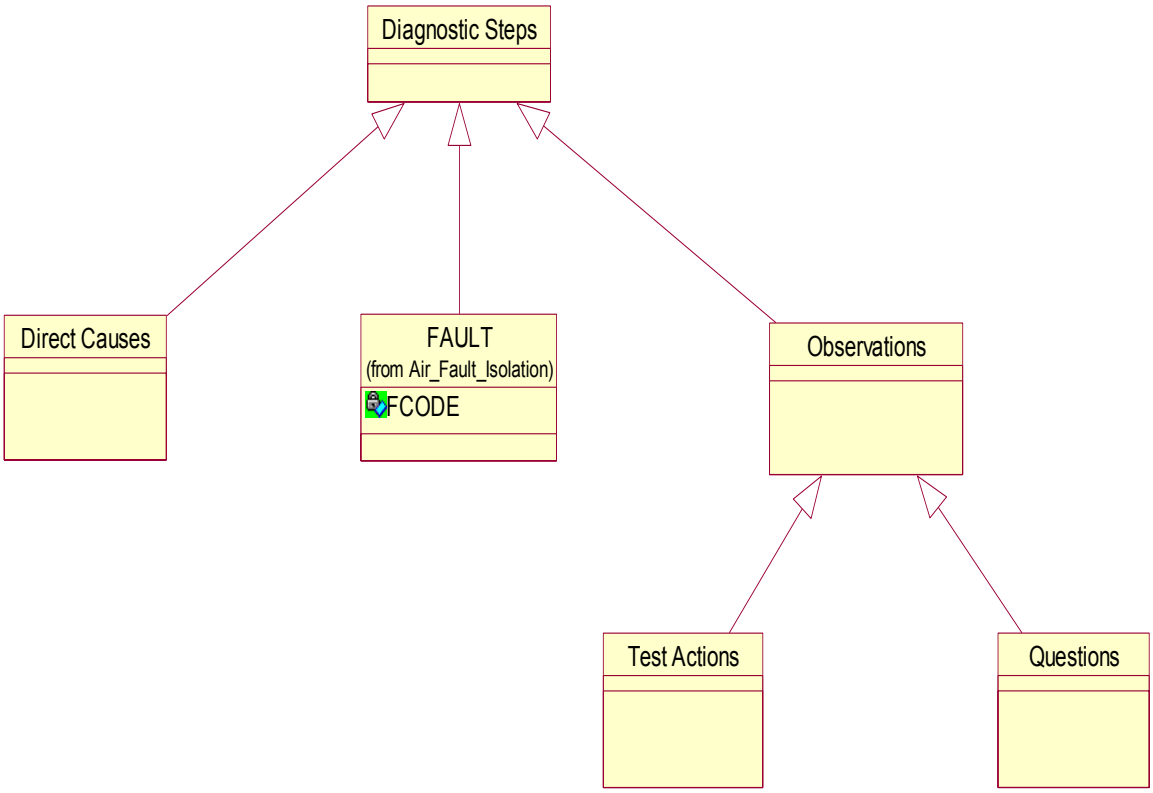


Figure 5.1: Diagnostic Steps Hierarchy

- *The fault*

This is the first step of the diagnostic procedure, which is the fault itself. Hence, this step establishes first that the fault is actually the same one that the technician is trying to troubleshoot. It is important to note that there is a maximum of one fault step in every diagnostic procedure. Hence if we are trying to fix multiple faults then there must exist multiple diagnostic procedures, one and only one for each fault (figure 5.2).

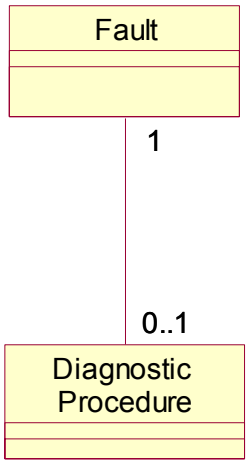


Figure 5.2: Fault to Diagnostic Procedures correspondence

In **Proc1** the fault would be:

- Car not starting

- *Direct causes*

These are steps that contain the direct cause of a specific fault. Ultimately, if the technician had determined correctly the root cause of a fault, he would have reached a step that contained a direct cause. It is important to point out that to each root cause is attached a repair procedure that gets rid of the root cause.

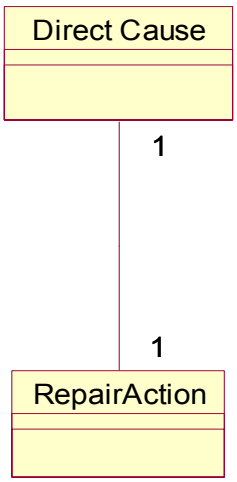


Figure 5.2b: Direct Cause to Repair Action Correspondence

In **Proc1**, the direct causes would be:

- Battery is dead
- Spark Plugs are dirty
- Car is out of fuel
- Fuel injection is dirty
- Leak in the fuel system
- Alarm is on

As we can see, any one of these causes or any combination of these causes can be the direct cause of the problem "car not starting".

- *Observations*

These can be of two kinds:

1. Questions: Sometimes answers to specific questions can shed light on the root cause of a particular fault. Questions in **Proc1** are:
    - Does the starter work?
    - Is there fuel in the car?
    - Are spark plugs clean?
    - Is the alarm on?
2. Test Actions: Actions that are performed, and there results will shed light on a specific cause. It is pertinent to say that performing test actions do not effectively change the state of the system, i.e. they do not fix the fault, they are just tests that are performed.

Test actions in **Proc1** are:

- Check Fuel gauge
- Press the "Alarm off" button
- Check Battery Plugs

Now that we have identified the several kinds of steps present in a diagnostic procedure, we shall present a technique for representing both observations and direct causes. We shall start with the representation of direct causes.

5.2.4 Representing direct causes

One way of structuring a situation where we have reasoning under uncertainty like the situation we have in **Proc1** is to construct a graph representing causal relations between events. A causal network consists of a set of variables and a set of directed edges, mathematically the structure is called a directed graph. When talking about relations within a directed graph, we use the wording of family relations: if there is a link from A to B, we say that B is a child of A and A is a parent of B [figure 5.3].

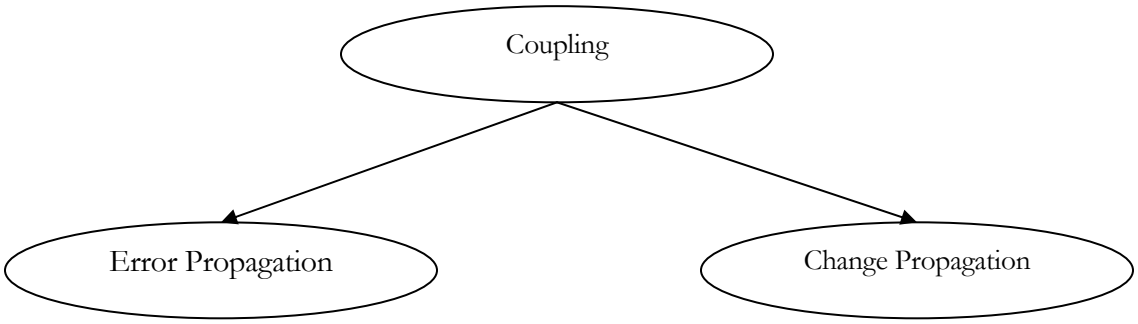


Figure 5.3: A causal network

In figure 5.3, we say that "coupling" is the parent, while the other two variables are the children. The variables in a causal network represent events or propositions. A variable

can have a number of states. For example, in figure 5.3 we have a causal network whose variables are:

- Coupling
- Error Propagation
- Change Propagation

Thus figure 5.3 depicts the relationships that might exist between coupling in software architecture and error propagation in that architecture as well as change propagation in the same architecture. Thus, we see that coupling might effect error propagation and change propagation. Each one of these variables has one or many states. For example, in the causal diagram above, the coupling variable might have two states: {High, Low}, while error propagation could have numeric states that represent ranges, for example {<0.65, >0.65}, the same goes for the change propagation variable.

As we can see from figure 5.3, "coupling" has influence on the other two variables. So any evidence on "coupling" will influence the certainty of the other two variables being in any of their corresponding states.

The situation in figure 5.3 is called a diverging connection [14]. Influence can pass between all the children of "coupling" unless the state of "coupling" is known. Thus if we don't know the value of "coupling", then observing the value of the "error propagation" variable will shed some evidence on "coupling", which in turn will effect our belief that the "change propagation" variable is in any of its states. On the other hand, if we know the exact value of "coupling", then the values of the other two children variables will not affect each other.

Causal relations also have a quantitative side, namely their strength that is expressed by attaching numbers to the links. Let ***A*** be the parent of ***B***. Using probability calculus [25], it would be natural to let **P (*B*| *A*)** (the probability of *B* given *A)* be the strength of the link. One kind of belief network that formalizes this relationship is a Bayesian belief network (refer to chapter 2 for more details). We shall use Bayesian belief

networks to model our fault procedure, and in particular the direct causes of a fault (figure 5.4). Thus we have the fault as the parent node, and all the direct causes as children of that node. So we see that we have to specify probabilities for each child node given its parent. For example, we have to specify P (Battery|Car Start), P (Plugs|Car Start), etc. What this means is that, given the fault has occurred what is the probability that Battery is the cause of the fault, that Plugs is the fault, etc. So we see that the sum of all the conditional probabilities across the children node must some up to one, that is:

$$\sum_{i=1}^{n} \mathbf{P\,(Ci \mid Fault)\ =\ 1}$$

Where *C* is a cause that is a child of the parent "fault" variable, and *n* is the number of children variables (figure 5.4). This is essential to our approach, hence the probability distribution must and should be exhaustive.

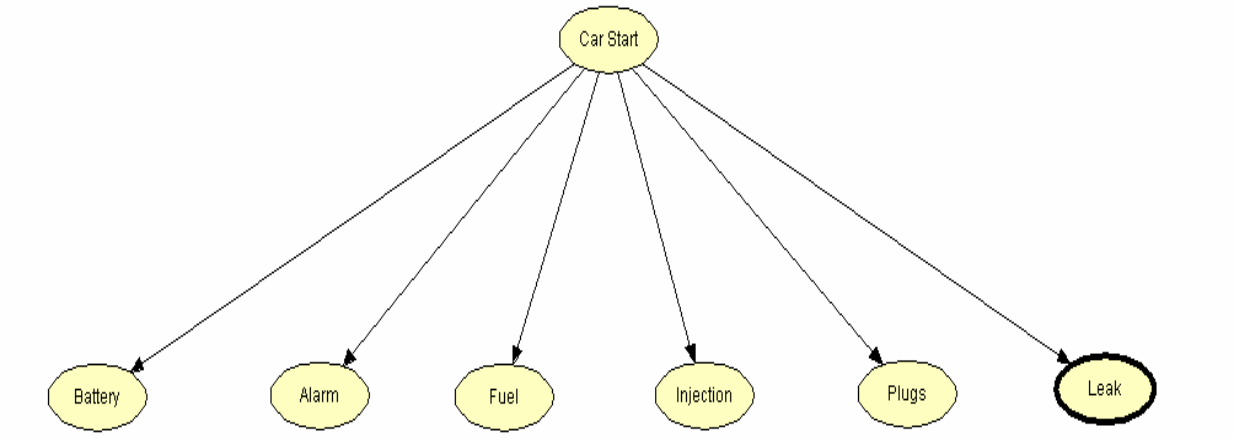


Figure 5.4: Bayesian belief network representing the direct causes in **Proc 1**

There will be two states for each child variable, these are:

1. FIX: Given that the fault has happened, and this cause is the actual cause of the fault, what is the probability that performing the repair action will fix this cause?
2. Not_FIX: The probability that performing the repair action will not fix this fault, given that the fault has happened and the cause is the actual cause of the fault.

So the summation of the probabilities of these two states must be equal to 1, such that:

$$P(Not\_Fix \mid Fault) = 1 - P(Fix \mid Fault)$$

Remember that the children of the fault node represent causes of that fault. So our ultimate aim is to fix that fault by repairing or removing the cause that caused it. In this light we say that the number of states in the fault node corresponds to the number of child variables this fault has. Each state represents the probability that the corresponding cause is actually the cause of the fault we are trying to diagnose (figure 5.5).

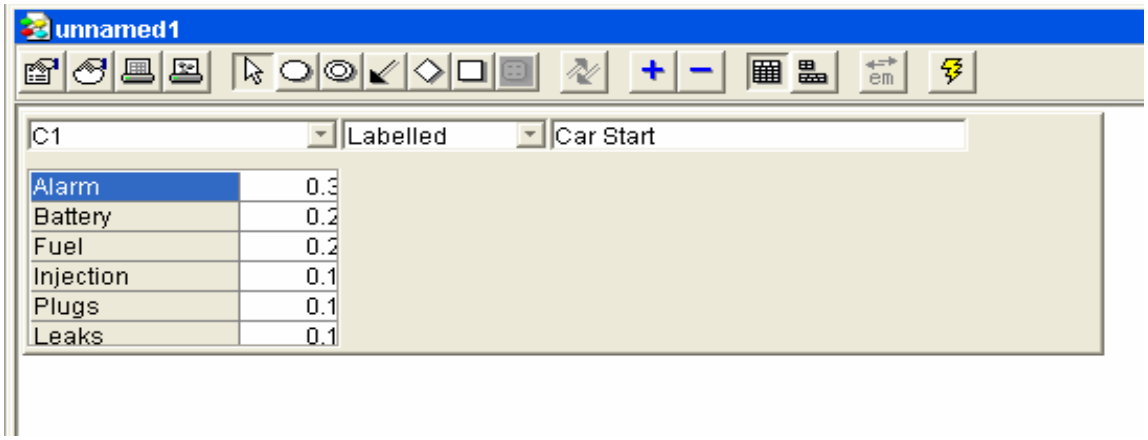


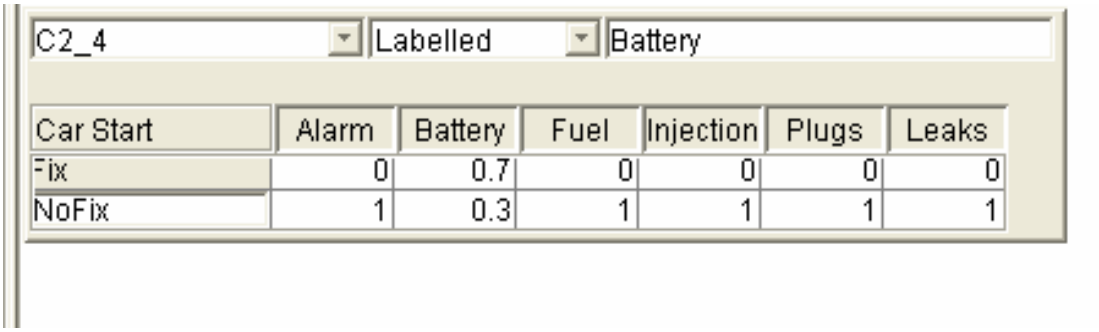
Figure 5.5: Probability table for the "Car Start" fault variable

The probabilities in figure 5.5 can be interpreted as follows:

- The Probability that the "Alarm" is on, is the cause that the car wont start is 30% (0.3).

- The Probability that the "Battery" is empty is the cause that the car wont start is 20% (0.3).

The same reasoning goes for "fuel", "injection", "Plugs" and "Leaks". You can notice that the summation of the probabilities must be equal to 1.



C2_4 | Labelled | Battery

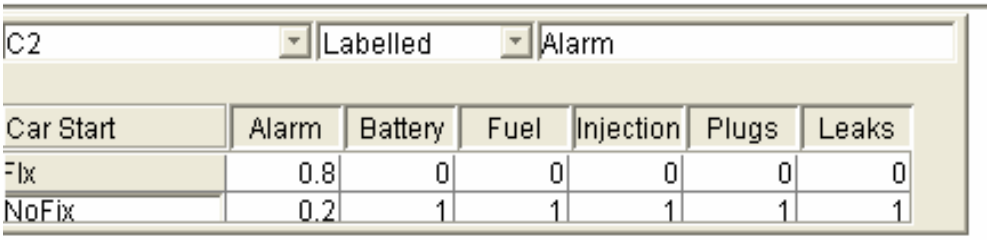
Car Start	Alarm	Battery	Fuel	Injection	Plugs	Leaks
Fix	0	0.7	0	0	0	0
NoFix	1	0.3	1	1	1	1

Figure 5.6: Probability table for the "Battery" variable

Figure 5.6 represents the probability table for the battery direct cause variable; the values inside the table are interpreted as follows:

- The probability that performing the repair action that repairs the battery problem, actually repairs this cause is 70% (0.7). The probability that it will not is 30% (0.3).
- The probability that this repair action will fix any other cause is 0.
- The probability that this repair action will not fix any other cause is 100 % (1).

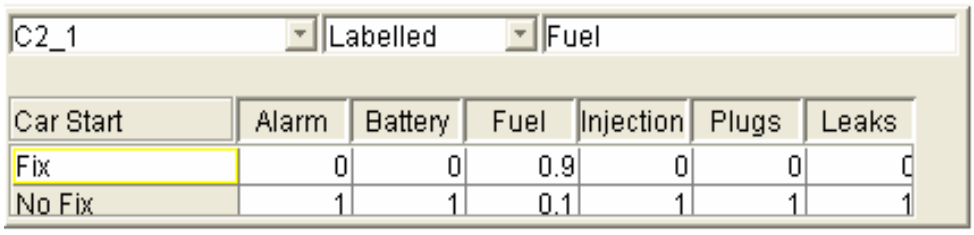
The same reasoning goes for figure 5.7, 5.8-11



C2 | Labelled | Alarm

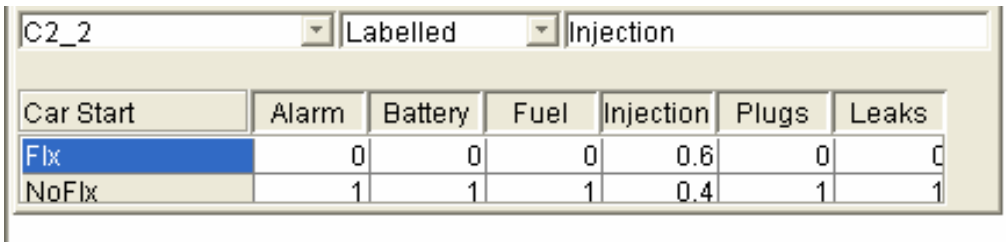
Car Start	Alarm	Battery	Fuel	Injection	Plugs	Leaks
Fix	0.8	0	0	0	0	0
NoFix	0.2	1	1	1	1	1

Figure 5.7: Probability table for the "Alarm" variable



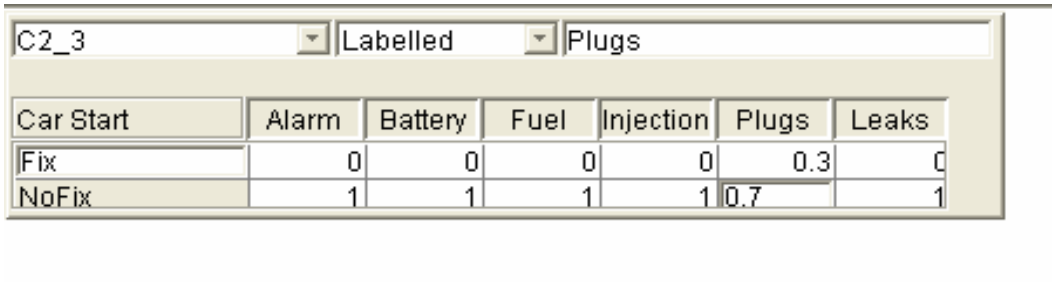
C2_1	Labelled	Fuel				
Car Start	Alarm	Battery	Fuel	Injection	Plugs	Leaks
Fix	0	0	0.9	0	0	0
No Fix	1	1	0.1	1	1	1

Figure 5.8: Probability table for the "Fuel" variable



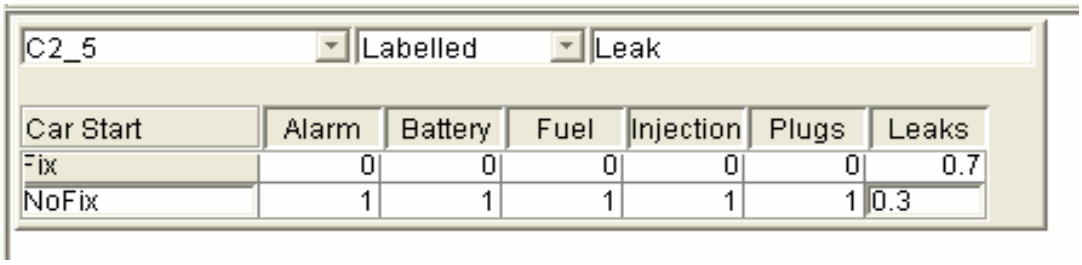
C2_2	Labelled	Injection				
Car Start	Alarm	Battery	Fuel	Injection	Plugs	Leaks
Fix	0	0	0	0.6	0	0
NoFix	1	1	1	0.4	1	1

Figure 5.9: Probability table for the "Injection" variable



C2_3	Labelled	Plugs				
Car Start	Alarm	Battery	Fuel	Injection	Plugs	Leaks
Fix	0	0	0	0	0.3	0
NoFix	1	1	1	1	0.7	1

Figure 5.10: Probability table for "Plugs" variable



C2_5	Labelled	Leak				
Car Start	Alarm	Battery	Fuel	Injection	Plugs	Leaks
Fix	0	0	0	0	0	0.7
NoFix	1	1	1	1	1	0.3

Figure 5.11: Probability table for the "leak" variable

5.2.5 Evidence Propagation

We will use the "Sum Normal" method for evidence propagation. This technique is presented in [26], and we will not discuss the details of the technique. The sum normal method is the most commonly used propagation method. It updates all probabilities, distribution functions, and expected utilities of the variables respectively, according to entered evidence.

By "evidence insertion" we mean that we observe that a certain variable is in its one of its states. To illustrate this, we will use the built in functions of the Hugin Expert tool [26].

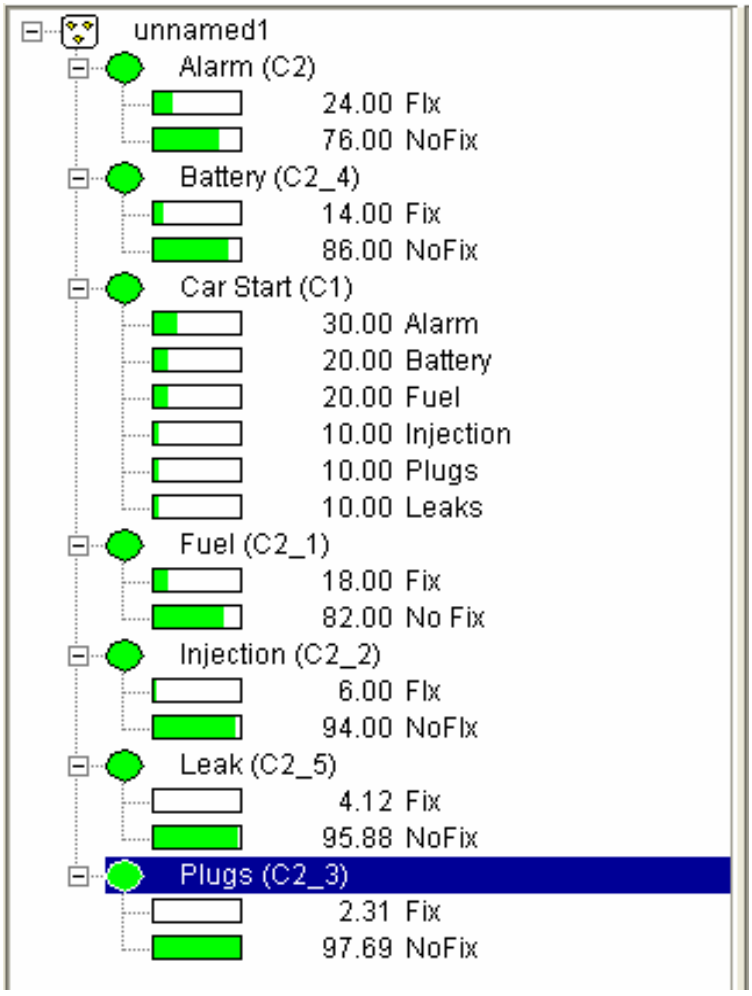


Figure 5.12: Propagation Results in the **Proc1** network

Figure 5.12 illustrates the initial propagation results of **Proc1**. As we can see, the results are consistent with the data that we input in the tables above. We can see that "Alarm" is the most probable cause of the fault. Later we shall develop a methodology that selects the most appropriate cause to select next to present to the user, based on more than just the above probabilities. Assume now that the user has observed that the "Alarm" was not the cause of the fault, so we must insert evidence into the network; that is variable "Alarm" is in "No Fix" state. The result is displayed in figure 5.13

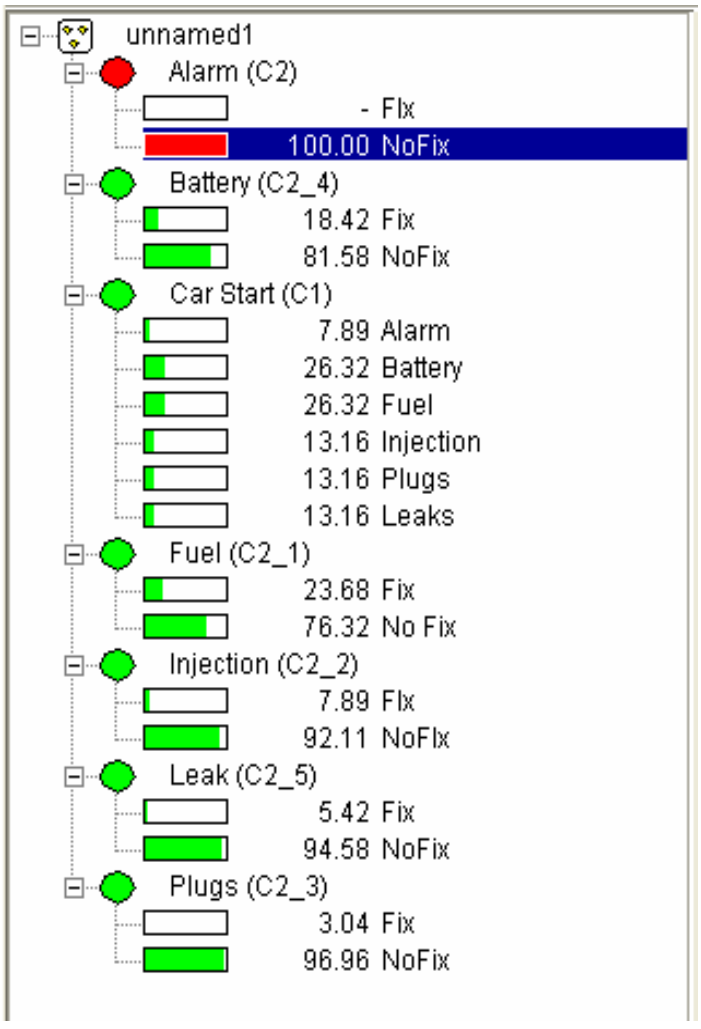


Figure 5.13: Propagation after observed evidence is entered

As we can see, the probability of the "Alarm" Variable has decreased, and the others have increased, which is consistent with our data.

As we have noticed from the above discussion, the Bayesian network approach relies heavily on the presence of previous data, and especially data elicited from experienced users, or documented manuals, such as IETMs.

Hence, the information we need to construct the Bayesian Network is:

- All the probabilities of causes/faults (Not Observations), these can be obtained from manuals in IETMs, or from experienced users, that explicitly enter the information in the network.
- All possible causes of the certain fault. This is essential; the network cannot predict a cause for a specific fault, if the cause was not incorporated in the network in the first place.
- All possible observations that a user can make that affect the fault. We will present a technique to represent observations in the system in the next section.

Moreover, the benefit of using Bayesian belief network approach lies in the following:

1. Can store information obtained from experienced users/manuals. This information is the probabilities presented in the above tables.
2. Evidence can be incorporated every time we have a new observation that a specific cause is not the cause of the fault. This is an essential property that we need, since evidence is continually being observed by the user, and we need to take that evidence into consideration when proposing to the user a diagnostic step to perform.
3. It can be expanded to incorporate as many causes as necessary. This is also essential to our effort, since the system can be updated every time a possible cause is discovered, enhancing the network's ability to better predict the cause of a fault.
4. Propagation methods are well established and statistically proven.

5.2.6 Representing Observations

As we have already discussed, a diagnostic procedure is composed of three main diagnostic step categories. The outcome of an observation may shed light on any of the possible faults. The troubleshooting task is to interleave actions and questions, such that the resulting sequence is optimal with regards to a specific person. In this section we will deal with the representation of observations within our framework. Unlike [16], we chose not to represent observations within the Bayesian network structure presented above. This is mainly for the following reasons:

- The Bayesian network structure gets more complex as more nodes are added. Probability tables can grow to be extremely large and incomprehensible.
- The Bayesian network structure needs, as stated above, information. In an IETM and many other environments, such information is simply not available or is cumbersome and expensive to get.

We need an approach that is simple, and incorporates as much of the information present already in the IETM. Our approach must also be consistent with IETM standards outlined in chapter 2. It must be portable, and consistent with open standards, so that it doesn't end up being too specific to be applied or ported to other applications.

Our approach is to model the observations in an XML format. The DTD is represented in figure 5.14. We chose XML, mainly because of its portability, and its unparalleled ability of explaining the meaning of data present in it by means of tags (refer to chapter 2).

As we can see from figure 5.14, an observation is made up of three parts. These are:

1. Question: Asked to a user, usually to make an observation about some evidence that could be incorporated in the network presented above.

2. Test action: An action that the user must perform to test the status of a specific part or meter in the system that the fault we are trying to diagnose lies in.
3. Answers: Possible answers the users must choose from when asked a question or when asked to perform a test action.

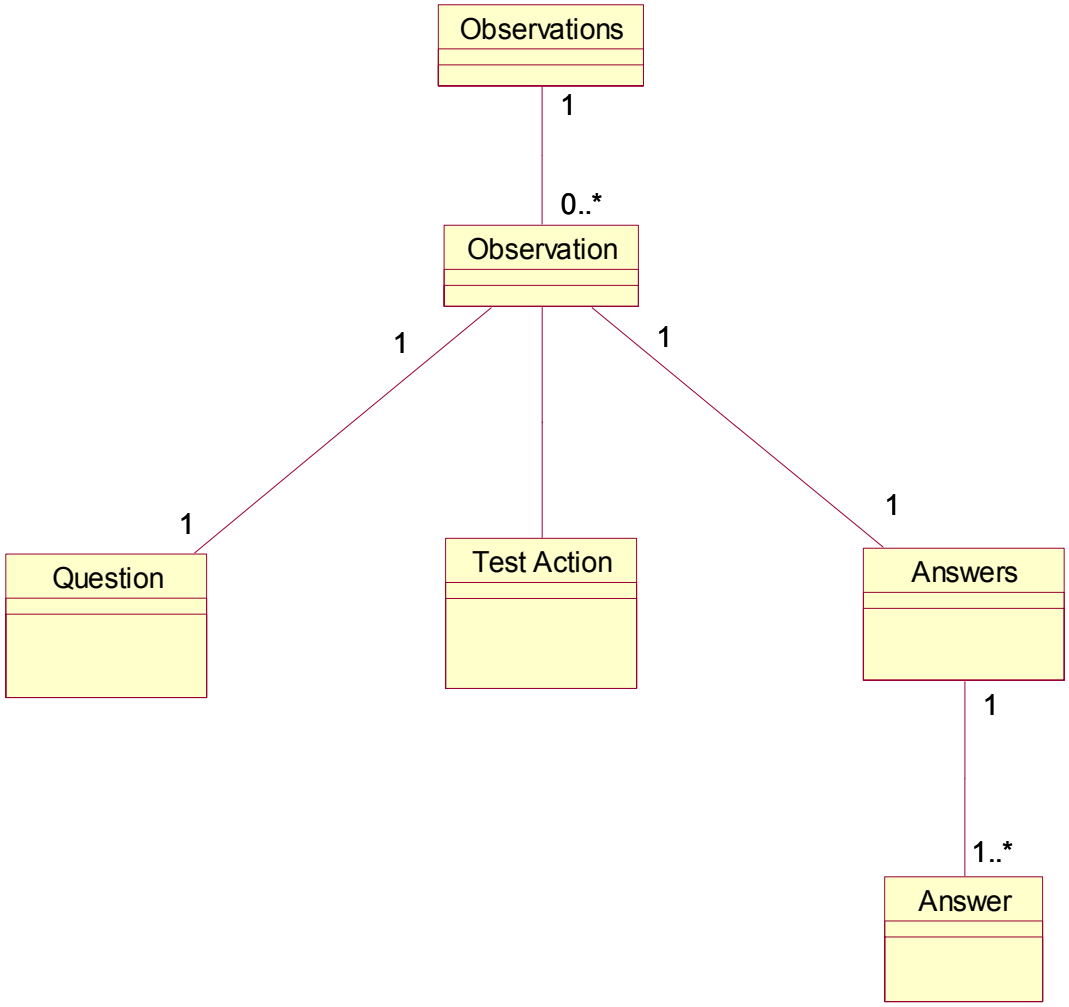


Figure 5.14: Structure of observations

As an example the XML document for **Proc 1** is depicted in figure 5.15. As we can see, XML is structured in a hierarchal organization, which makes it ideal to structure our data. In it is also prudent to point out that we have used the "relaxed DTD" mapping (refer to chapter 2 for more details) to represent the UML diagram presented in figure 5.14, in the XML format presented in figure 5.15

```
- <observations>
  - <observation>
      <question>Is the Fuel Indicator on Empty?</question>
    - <answers>
        <answer>Yes</answer>
        <answer>No</answer>
      </answers>
    </observation>
  - <observation>
      <question>Do you hear the starter roll?</question>
    - <answers>
        <answer>Yes</answer>
        <answer>No</answer>
      </answers>
    </observation>
  - <observation>
      <question>Is the alarm light on?</question>
    - <answers>
        <answer>Yes</answer>
        <answer>No</answer>
      </answers>
    </observation>
  - <observation>
      <question>Are the spark plugs clean?</question>
    - <answers>
        <answer>Yes</answer>
        <answer>No</answer>
      </answers>
    </observation>
  </observations>
```

Figure 5.15: Observation organization in **Proc 1**

5.2.7 Adaptation Policies

4.2.7.1 Overview

As stated before, the troubleshooting task is to interleave actions that a user must do and observations that a user must make, to achieve an optimal sequence. Now, an optimal sequence, with regards to our goal, is relative to a specific user. Thus, a novice user may prefer a sequence that is different that may be not as efficient (has higher probability of fixing the fault) as a sequence that an expert user might prefer, but is relatively easier to do.

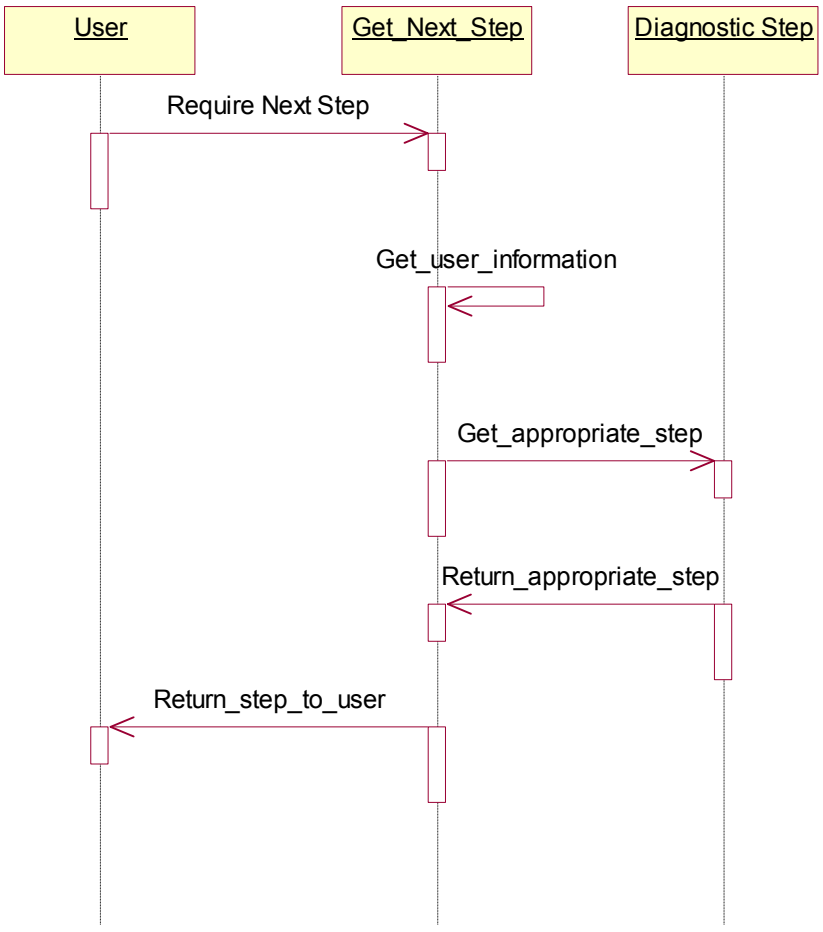


Figure 5.16: Sequence diagram for diagnostic step request

As we can see from figure 5.16, getting the appropriate next step is not always the same for every user. According to the information that the user provides about himself, the next step to display will be determined by our system, thus the sequence always is "reorganized" according to the user requesting the diagnostic procedure.

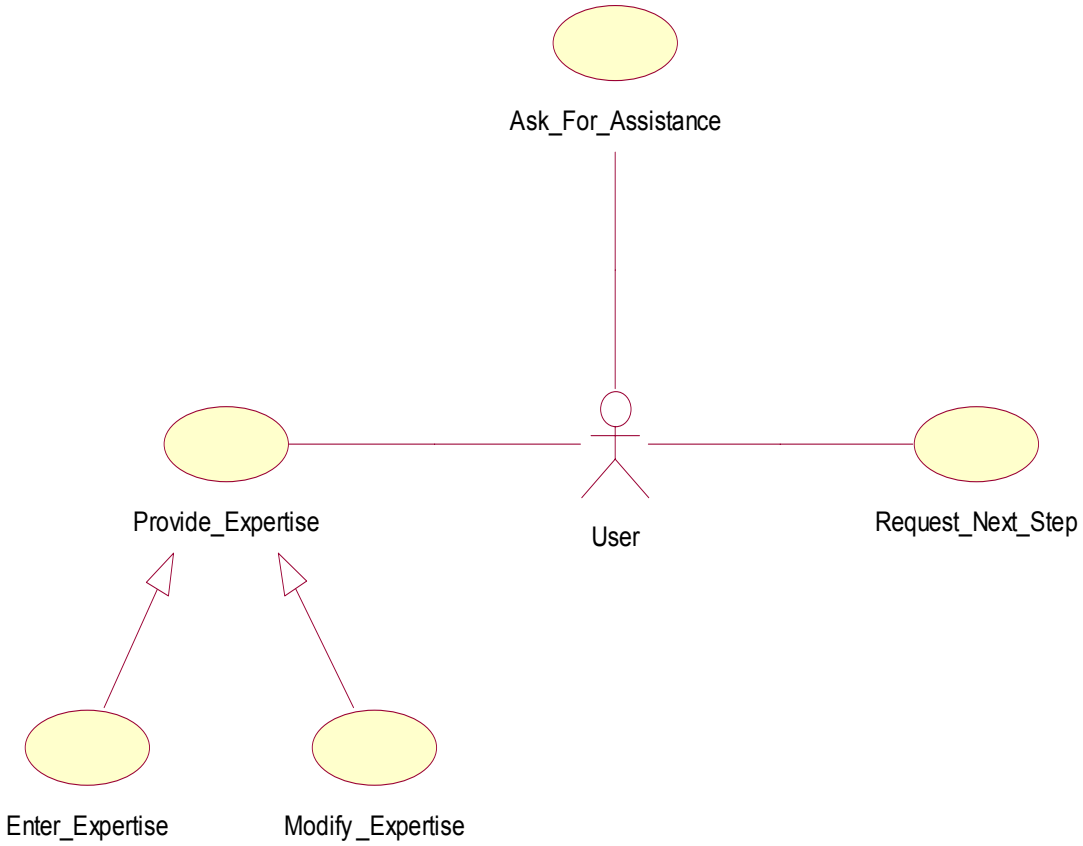


Figure 5.17: User actions in a diagnostic procedure

As we can see from figure 5.17, a user can basically do three actions when requesting or performing a diagnostic procedure.

Thus, the user can:

1. Provide his expertise, which is further divided into:
    - Entering the expertise upon initial request of the procedure

- Modifying his expertise while performing the procedure

2. Request next step: the user can request from the system the next step in a diagnostic procedure to be displayed.
3. Ask for assistance: If the user finds difficulty in performing a particular action, the user might request assistance or more information to be displayed about performing this particular action.

4.2.7.2 Preliminaries

Recall from the previous sections, that the system must be able to adapt to several factors, including the user's expertise, the step's difficulty as well as the system's prior history. In this light, let us define the following:

- A cost **C**, attached to each and every diagnostic step in a procedure. This cost represents the difficulty of the step performed. This measure, as stated earlier is subjective and given by experts, either directly or documented in an IETM. The cost is a range between 1 and 10. The higher the cost of a particular step the harder it is to be performed, and the higher the level of expertise required to be performed correctly. In measuring the cost of step factors such as the number of tool needed to perform the action, their complexity, and the time required to perform the repair should be taken into consideration.
- An experience level **E** which denotes the level of expertise of the user. This number is a range between 1 and 10. The higher the number, the more experienced the user. This number is related to the cost C defined above, in the sense that if a step has cost less that the experience level of the user, it is easier for him to do than a step which has a cost greater than his experience.
- As stated before, with each cause a probability **P** is attached. It is important to say that **P** here refers to the values in the network after it has been propagated.

To remove the burden of structuring the Bayesian network discussed above, and to better represent the values outlined (**C**,**E** and **P**), we use XML to represent the network, and then automatically construct the network, and propagate any evidence in it (figure 5.18).

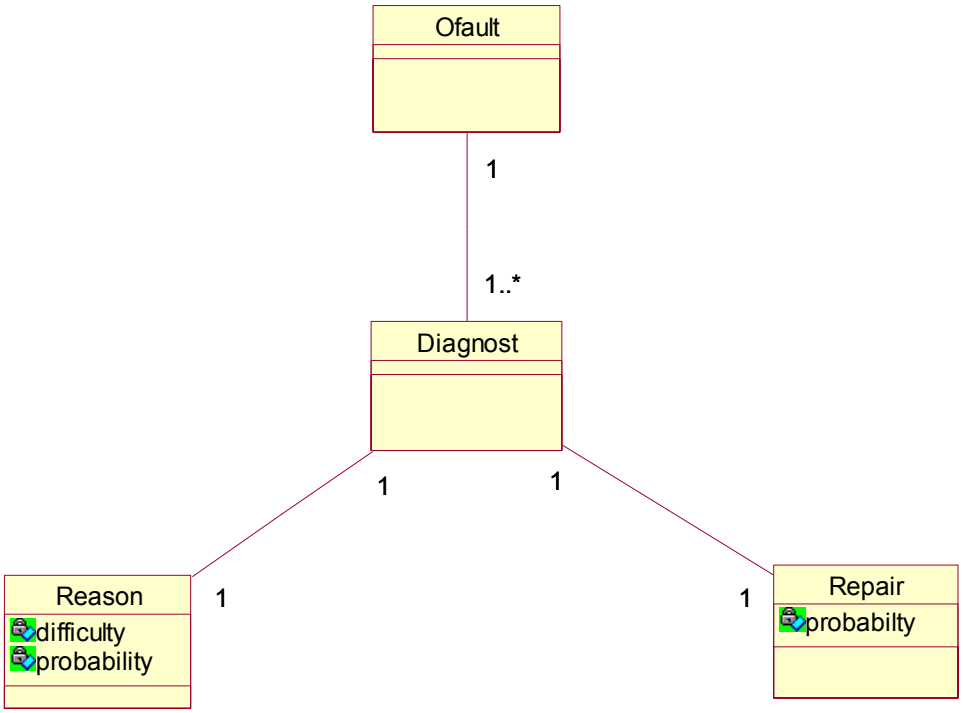


Figure 5.18: Structure of direct causes

As we can see in figure 5.18, we have:

- Ofault: represents the observed fault we are trying to diagnose.
- Diagnost: a diagnostic step, this, in turn has two children:
    - Reason: an actual cause of a fault, attached to it is the probability, as discussed earlier and the difficulty of it being repaired (**C**).
    - Repair: the repair action, that when performed fixes the cause. It also has a probability attached, as discussed earlier.

As an example, figure 5.19 depicts how causes are documented in **Proc 1**:

```
- <ofault>
  - <diagnost>
      <reason difficulty="3" probability="0.3">Alarm is on</reason>
      <repair probabilty="0.8">Press the "Alarm Off" button</repair>
    </diagnost>
  - <diagnost>
      <reason difficulty="6" probability="0.2">Battery is empty</reason>
      <repair probabilty="0.7">Recharge/Replace Battery</repair>
    </diagnost>
  - <diagnost>
      <reason difficulty="1" probability="0.2">Empty Fuel Tank</reason>
      <repair probabilty="0.9">Put Fuel in the tank</repair>
    </diagnost>
  - <diagnost>
      <reason difficulty="9" probability="0.1">Fuel Injection clugged up</reason>
      <repair probabilty="0.6">Clean the injection nozzles</repair>
    </diagnost>
  - <diagnost>
      <reason difficulty="6" probability="0.1">Dirty Spark Plugs</reason>
      <repair probabilty="0.3">Change/Clean Spark Plugs</repair>
    </diagnost>
  - <diagnost>
      <reason difficulty="8" probability="0.1">Injection Leak</reason>
      <repair probabilty="0.7">Check fuel hoses</repair>
    </diagnost>
  </ofault>
```

Figure 5.19: XML structure representing the values **C** and **P**

In addition to the above we define the following:

- A utility **U** for each step. U is defined as a ratio of the user experience to the cost of the step, thus:

$$U(step) = \frac{E(user)}{C(step)} \qquad (1)$$

Obviously, the higher the utility of a diagnostic step, the more appropriate it is to the user. This is because we want a step with the lowest cost, hence the higher the cost of a step, the lower the utility of a step. Moreover, the higher the expertise of the user, the higher the utility of the step will be. So we see that as the difference of between the experience of the user and the cost of the step increases, in favor of the experience, i.e. as **E-C** gets greater than one, the higher the utility of step will be.

In (1) we have related the experience of the user with the level of difficulty of the step. However, this is not enough to our goal, for we must also take the probability of the step we discussed in the previous section. So we define the following:

• A weight **W** attached to each step. W is defined as the product of the utility defined in (1) with the probability of the cause after we propagate the network (discussed earlier). Hence **W** is defined as:

$$W(step) = U(step) \times P(step) \qquad (2)$$

Hence, like **U,** the higher the higher the weight, the more the step is appropriate for a specific user. What we are saying here is that as the cost of the step decreases and its

probability of fixing the fault increases, and the users experience increases, the better the step is with regards to the user.

However (1) and (2), don't take the following intuitive considerations into account:

- If the user has a higher expertise than is required for the step (as discussed earlier), (**E**>**C**) the utility shouldn't matter as much as the probability. The reasoning behind that is that if a user is an expert, then we should give more weight to the step which has the higher probability of fixing the fault. Hence equation (3) is now of the form:

$$W(step) = U(step) \times P(step) + P(step) \qquad (2b)$$

  It is important to say that during execution of our case study, equation (2b) gave better results that (2) alone in the case where (E>C).

- On the contrary, if the user has lower expertise than is required for the step (**E**<**C**), we should take good consideration that we don't present to the user a task that is impossible to him to perform correctly, even though its probability of fixing the fault is higher than other step. Thus we must take careful consideration that we give more importance to the difference between the step's cost and the user's expertise, hence (2) becomes:

$$W(step) = U(step) \times P(step) + \frac{E(user) - C(step)}{100} \qquad (2c)$$

Hence we as **C** gets greater and greater than **E**, we are decreasing the weight of the step (E-C increases in the negative direction). In the following chapter we shall examine a case study that illustrates all the preceding formulas.

4.2.7.3 Adapting the system

Now that we have defined all the necessary terms, we shall present a methodology for selecting the next step to display to the user, in a diagnostic procedure. The methodology goes as follows:

1. Build the Bayesian network that contains the causes (as explained earlier).
2. Any Evidence, if present, is entered into the system. For example, we have observed that a certain variable is in a particular state (as explained earlier).
3. The system is propagated using the propagation method discussed earlier.
4. The new probabilities (**P**) are extracted (from the network).
5. **W** and **U** of a cause are calculated, according to (1), (2b) or (2c).
6. The Node with the highest weight **W** is presented to the user

As we will show in the following chapter, since **U** depends on **E**, then **W** varies for each user, hence, a different order of steps is presented.

Recall that as the user navigates through the system, he is continually presented with steps that he must perform. Some of these steps have costs that are higher than his expertise, since sometimes we can't avoid doing that, and others have costs that are less than the user's expert level. Our methodology takes into account the above considerations and continually monitors and adjusts the user's experience according to his performance while executing the diagnostic procedure. Thus, we must always adjust the user's expertise level to suggest more appropriate steps that fit his expertise level. Hence, a user may enter an expertise level at system start up, but while he navigates through the diagnostic procedure, we notice that he is more of an expert than he

claimed he is, then he is capable of performing more complex tasks that have higher probabilities of fixing the fault. On the other hand, we might find out that the user is less of an expert than he claims he is, then we must lower his expertise level, for the same reason stated above.

In the light of the above situation we define the following rules to continuously adjust the user's expertise level:

- If the step proposed to the user has a cost **C** which is lower than the user's expertise **E** , (**E**>**C**) then:
    - If the step is done correctly, then no change. This is because we expect a user which has a higher expertise to perform the step correctly.
    - If the step is not done correctly, then we must decrease his level of expertise by:

$$1 - \frac{1}{U} \qquad (3)$$

Since U = E/C, the higher the difference between E and C, the higher the decrease will be.

Then the new expertise will be:

$$E(new) = E(old) - \left(1 - \frac{1}{U}\right) \qquad (4)$$

That means that if a user doesn't perform correctly a step that has cost much less than his expertise, we must decrease his expertise by a higher factor. For example, if the E=6 and C=5, (3) will evaluate to: 0.17, while if C = 1, then it will evaluate to: 0.84, hence if the cost of a step is much lower than the expertise of a user, we expect him to correctly perform the procedure, if not, we must decrease his expertise by a larger factor.

- If the step proposed to the user has a cost **C** which is higher than the user's expertise **E** , (**E**<**C**) then:

    - If step not done correctly, then there is no change, since we expect a user which has expertise less than the cost of a step to have some difficulty in performing the repair action.
    - If the step is done correctly, then we must increase his expertise by:

$$\mathbf{E\,(new)\ =\ E\,(old)\ +\ 1\ -\ U} \qquad \mathbf{(5)}$$

The reasoning is the same as before. As the cost gets higher and the expertise gets lower, and the action is performed correctly, then the increase gets higher.

Now we will turn our focus to observations. Recall that observations are questions that are asked to the user to shed light on the cause of a fault. They are not direct causes themselves. We must find a way to see when to ask a user a question to shed light on a specific cause and when to give him the cause directly without bothering him with questions. Our goal is to determine the best observation-cause sequence to be presented to the user, and we remove the bottleneck of having manual writers

to explicitly code the importance of an observation to a cause. This is unacceptable, because the system, especially an IETM, can grow to be very large, and having the writers of the manual to explicitly tell us what is the effect of each question on each and every cause, is unacceptable, and considered as a bottleneck.

In this light, we shall use data mining techniques to see what question, if any, effects the weights of the steps at any point in the diagnostic procedure. In order to achieve this goal, we must have a large set of data which contain answers to questions that users have answered, and the ultimate cause they found at the time they answered the question. This is because we will try to find a pattern of specific answers to observations that may influence a specific cause. For example if we found that 100% of the time that users answered question 1 as "yes", the cause turned out to be a specific cause, then asking that question to the user, might shed light on weather that cause is the ultimate cause of the fault we are trying to diagnose. Thus, if the user answers "yes" to the question then, we will increase the weight of that cause.

Hence we shall use *funnel theory* [27] to "filter" or "funnel" out observations that most affect the weights of the causes, as discussed above. We will use the following formula based on [27]:

$$\mathbf{Ef}(a.r) = \frac{\mathbf{best}(a.r) - \mathbf{rest}(a.r)}{\mathbf{all}(a.r)} \qquad (7)$$

Where "a" is the observation and "r" is the answer of the observation. For example, in **Proc 1** and from figure 5.15, "**Is the fuel Indicator on Empty.Yes**" is an example "a.r". Hence (7) determines the effect (**Ef**) of an observation/answer pair (a.r) on a specific cause. Hence, "best" represents the cause we are trying to find the effect of "a.r" on. Then, best (a.r), is the number of times (a.r) appears in a diagnostic sequence

where the root cause was found to be "best". On the other hand "rest(a.r)", represents the appearance of "a.r" in the rest of the sequences where the rest of the causes where found out to be the cause, and all(a.r) represents the number of appearances of "a,r" in all of the diagnostic sequences.

Hence, we apply (7) in the following manner:

1. For each "**a.r**" calculate **Ef (a.r)** for every "**best**". That is, calculate the effect of every observation-answer pair on every cause.
2. Get the highest **Ef (a.r) = Max (Ef (a.r)).**
3. If the cause that **Max (Ef (a.r))** belongs to is different from the one that already has the highest weight, then this observation affect other classes, and might shift or change the weights of the causes, hence asking it to the user is pertinent.
4. If the users answers "**r**" to "**a**", then we add **Ef (a.r)** to the weight of the corresponding cause (to give it more importance).
5. Display the cause with the highest weight (as discussed earlier).

Hence in this manner we interleave observations and direct causes in any diagnostic sequence, avoiding asking questions that don't really shed any evidence on a specific cause.

The benefits of the above approach to handling observations is that at time of design, IETM authors don't have to specifically code the effect of each observation on a specific cause, because we are just mining answers to questions that are collected when users actually used the system, even though these users might be expert mechanics in their field, and might be "training" the system for other users, so that it has a large set of data to mine.

Part III: Implementation

In this section we present the implementation details and we go through a case study to present the methodology outlined above

CHAPTER 6: CASE STUDY AND IMPLEMENTATION

To determine the applicability of our proposed methodology, we implement a proof of concept demonstration system. By no means do we claim that we have developed a full fledged IETM system. IETM systems, as discussed earlier, have large and various components of different nature, depending on the specification these systems are implemented on. In our implementation, we focus on implementing the adaptive fault diagnosis methodology discussed in the previous chapter.

6.1 Fault Procedure

Before we indulge in the description of the architecture, we shall take an example fault procedure as we did in the previous chapter. Due to the highly technical nature of fault procedures present in IETMs, which mainly deals with complex mechanical systems, primarily of military nature, we will avoid in this demonstration these kinds of fault procedure. We shall, instead, focus on a more intuitive daily life situation involving a car leak in a car. The fault procedure has the following attributes:

- Fault: Liquid found under the hood of a car, i.e. "Car Leak"
- Causes that can be the root of the problem:

Causes	P(Cause)	P(Repair)	Cost of Repair
Fault Radiator Pump	0.5	0.75	6
Faulty Washer Pump	0.3	0.515	3
Faulty Oil Pump	0.1	0.6	1
Water Hose Leak	0.1	0.5	7

Table 6.1: Causes of the fault "Car Leak"

In table 6.1, we give each cause with its P (Cause), P (Repair) and Cost of repair, as discussed in the previous chapter.

- Observations of interest that a user might observe during the act of performing the diagnostic procedure:
    - Is there A Liquid under the Engine?
        - Possible answers: Yes/No
    - What is the Color of The Liquid?
        - Possible answers:
            - Blue
            - Black
            - Green
            - Clear
    - What Year was the car made?
        - Possible answers:
            - 1970-1980
            - 1980-1990
            - 1990 and Above
    - Was the Car Damaged In Some Way from the Front?
        - Possible answers: Yes/No

As we see we don't specifically put the effect of every observation/answer pair on each and every cause, because we are calculating **Ef** as discussed in the previous chapter.

Formally, the cause/observation steps are organized in an XML file as described in figure 5.1. Figure 6.1 depicts that file.

```
- <ofault name="Water Leak" xmlns="" number="code1794">
  - <diagnost number="code1795">
      <reason difficulty="5" probfac="0.4" number="code1796">Faulty Radiator Pump</reason>
    - <repair probfac="0.75" cost="6" number="code1797">
        <ref number="code1798">Referecnce for Radiator Pumps goes Here</ref>
      </repair>
    </diagnost>
  - <diagnost number="code1799">
      <reason difficulty="7" probfac="0.3" number="code1800">Faulty Washer Pump</reason>
    - <repair probfac="0.515" cost="3" number="code1801">
        <ref number="code1802">Referecnce for Washer goes Here</ref>
      </repair>
    </diagnost>
  - <diagnost number="code1819">
      <reason difficulty="5" probfac="0.2" number="code1820">Faulty Oil Pump</reason>
    - <repair probfac="0.6" cost="7" number="code1821">
        <ref number="code1822">Referecnce for Oil Pumps goes Here</ref>
      </repair>
    </diagnost>
  - <diagnost number="code1823">
      <reason difficulty="5" probfac="0.1" number="code1824">Water Hose Leak</reason>
    - <repair probfac="0.5" cost="7" number="code1825">
        <ref number="code1826">Referecnce Water Hose Leak Goes Here</ref>
      </repair>
    </diagnost>
  - <observation number="code1803">
      <question type="boolean" number="code1804">Is there A Liquid Under the Engine?</question>
    </observation>
  - <observation number="code1805">
      <question type="definite" number="code1806">What is the Color of The Liquid?</question>
    - <answers number="code1807">
        <answer number="code1808">Blue</answer>
        <answer number="code1809">Black</answer>
        <answer number="code1810">Green</answer>
        <answer number="code1811">Clear</answer>
      </answers>
    </observation>
  - <observation number="code1812">
      <question type="Range" number="code1813">What Year was the car made?</question>
    - <answers number="code1819">
        <answer number="code1814">1970-1980</answer>
        <answer number="code1815">1980-1990</answer>
        <answer number="code1816">1990 and Above</answer>
      </answers>
    </observation>
  - <observation number="code1817">
      <question type="boolean" number="code1818">Was the Car Damaged In Some Way from the Front?</question>
    </observation>
</ofault>
```

Figure 6.1: XML File of Fault Procedure

Now that we have established the fault procedure we will use in our case study, we shall present the architecture of the system.

6.2 System Architecture

As we have stated before, we are not designing a full fledged IETM system, however we will like our system to be expandable to one. That is why we chose a component-based architecture for the design of our system. These architectures have a very important characteristic which is that they are expandable to the highest degree, thus you can add components to the system that implement more or different functionalities, with little or no modification to the existing architecture. Figure 6.2 represents the top level architecture of our demonstration system, the description and functionality of the components is as follows:

- `Fault_Manager`: Responsible for Managing the retrieval, selection and presentation of fault diagnostic steps to the user.
- `Bayesian_Builder`: Responsible for Building the Bayesian Belief Network.
- `Observation_Handler:` Handles all observation related functionalities (as discussed in the previous chapter)
- `Retreival_Manager`: Responsible for the retrieval of fault diagnosis procedures from the IETM.
- `Display_Procedures`: Handles the display of data and GUI related elements to the user
- `Adapt_User`: Handles and implements the adaptation formulas presented in the previous chapter.
- `User_Handler`: Handles all inputs that the user enters.

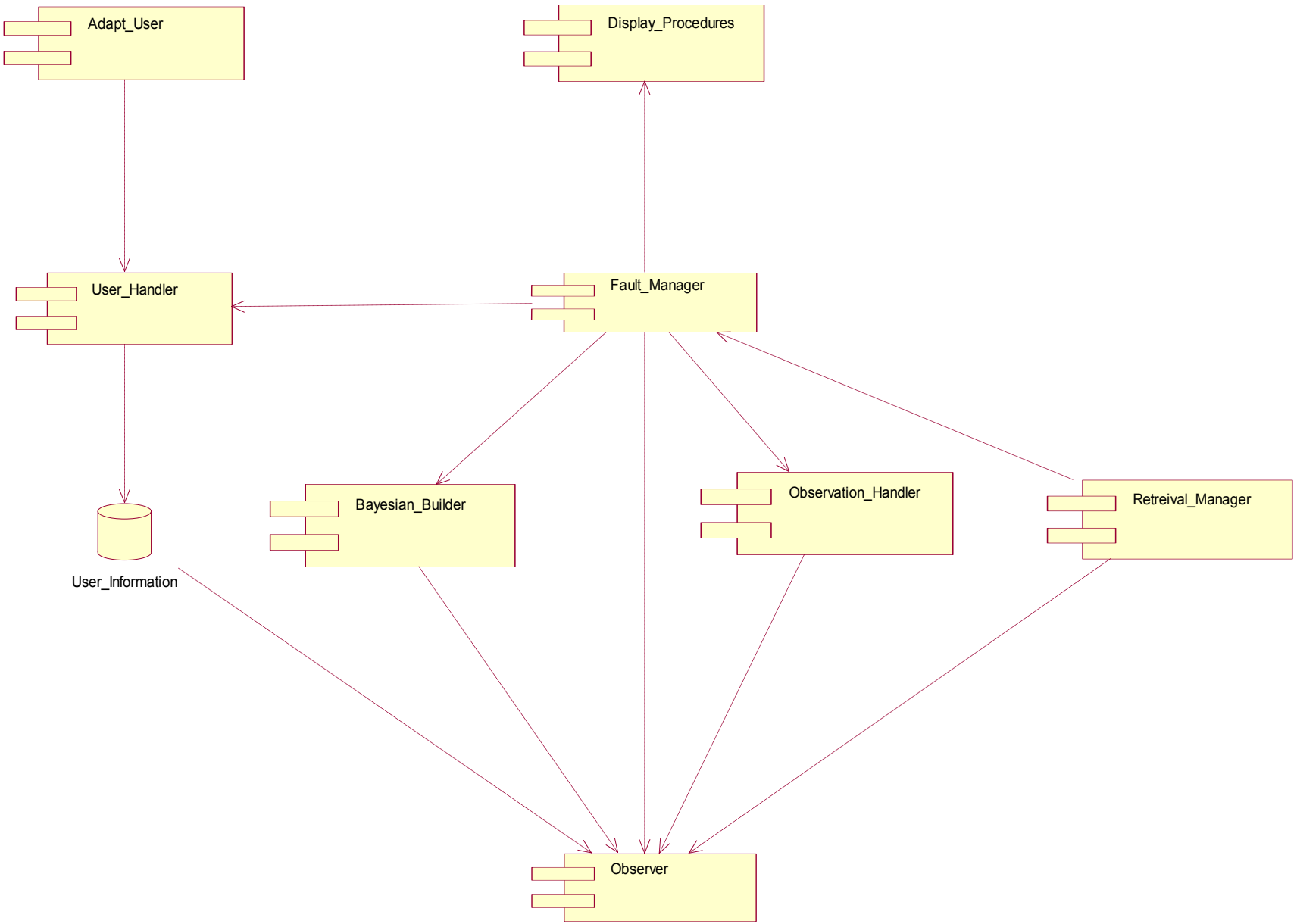


Figure 6.2: Top level system architecture

- `Observer`: This component is for our demonstration purpose only, and shows how the measures (Experience, Weights, etc.) change as the user navigates through the diagnostic procedure.

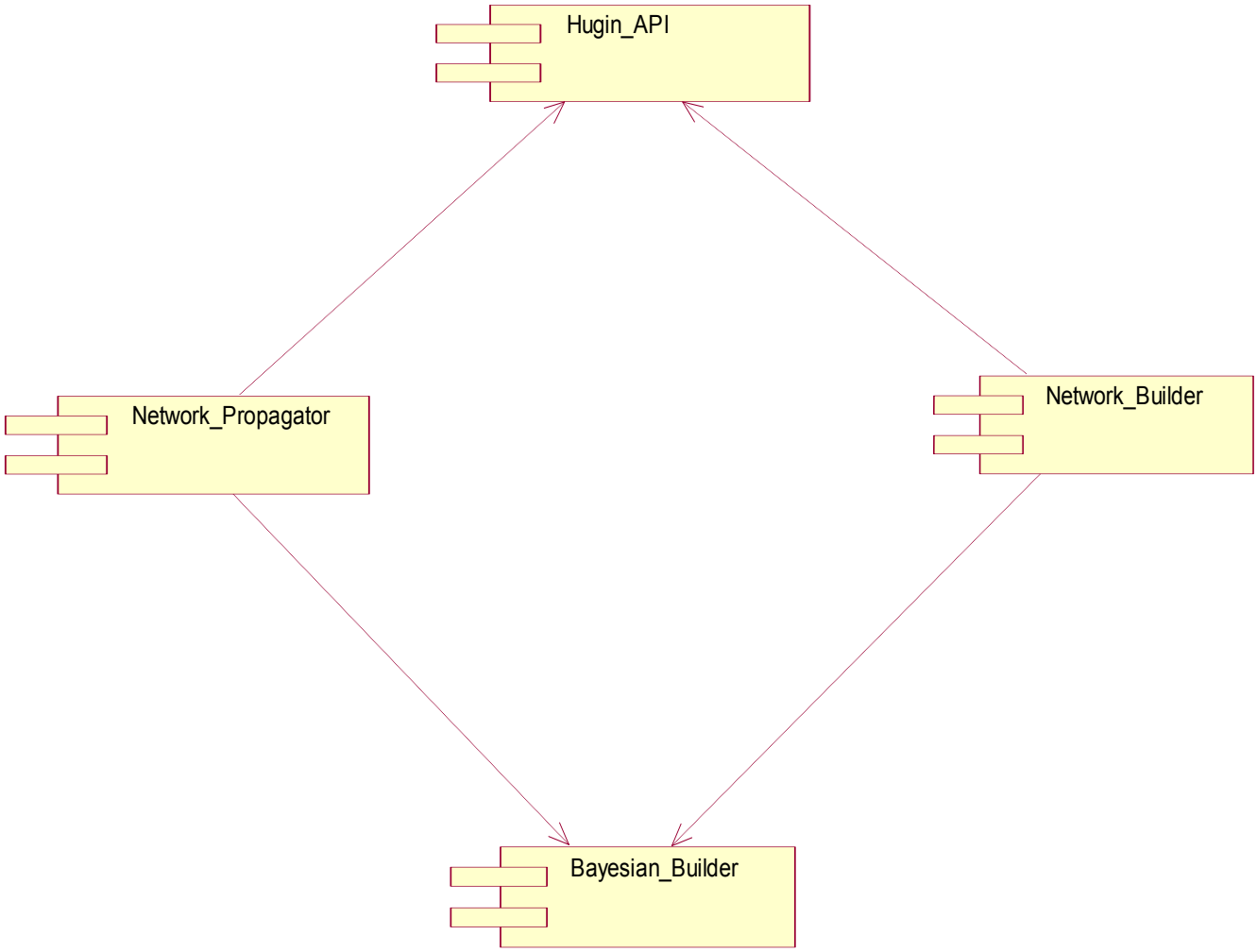


Figure 6.3: Bayesian_Builder Architecture

In turn, the `Bayesian_Builder` (figure 6.3) Component is composed of:

- `Network_Builder`: Responsible for building the structure of the network
- `Network_Propagator`: Responsible for propagating evidence inside the network.

- `Hugin_API`: Hugin Expert is a tool that has an exposed API, that allows you to build Bayesian Network. This Component provides the link between our system and the Hugin API.

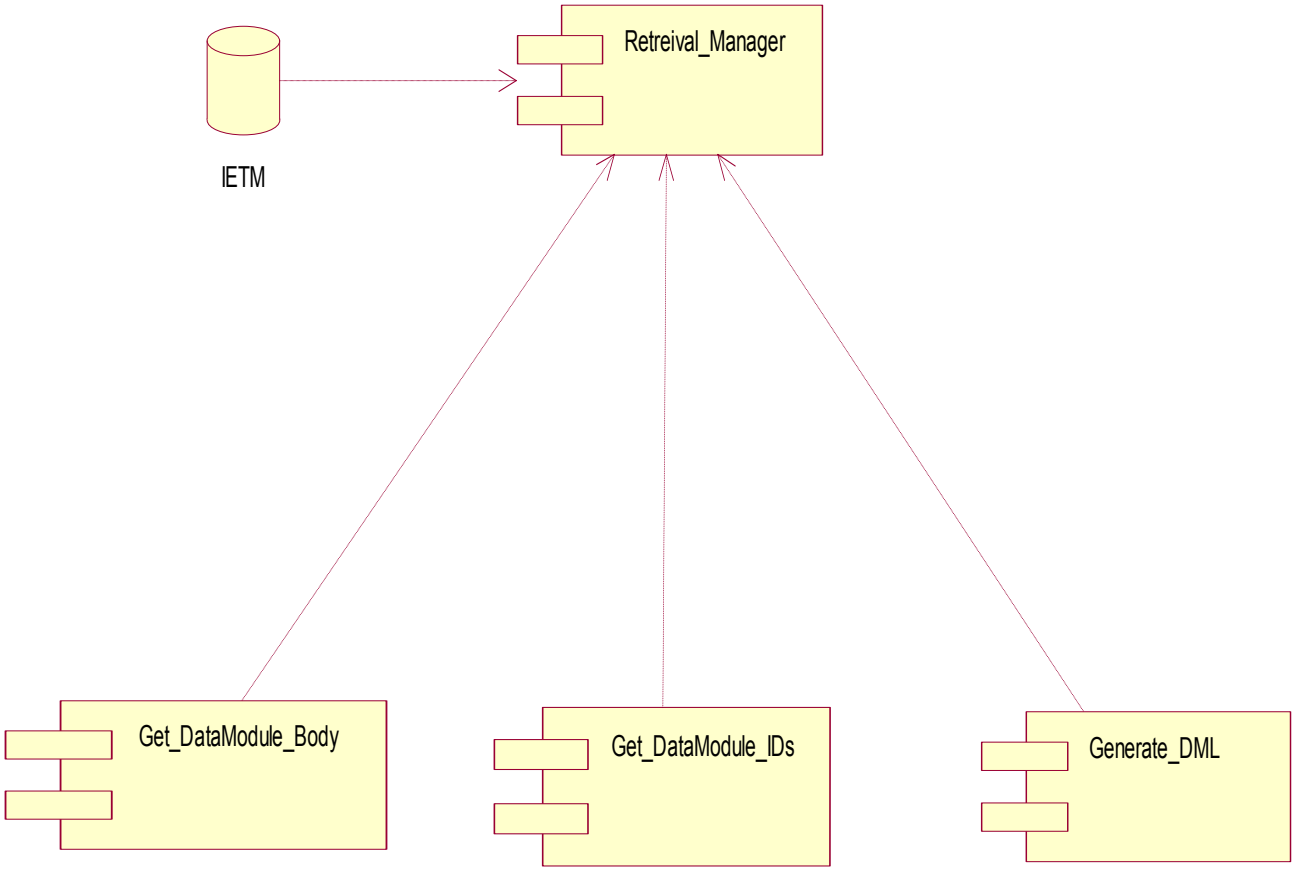


Figure 6.4: Retreival_Manager architecture

The `Retreival_Manager` component is composed of:

- `Get_DataModule_Body`: Responsible for retrieving data module XML bodies
- `Get_DataModule_IDs`: Responsible for getting and managing data module ID (as explained in chapter 4).
- `Generate_DML`: Generates lists of available data modules in an IETM CSDB.

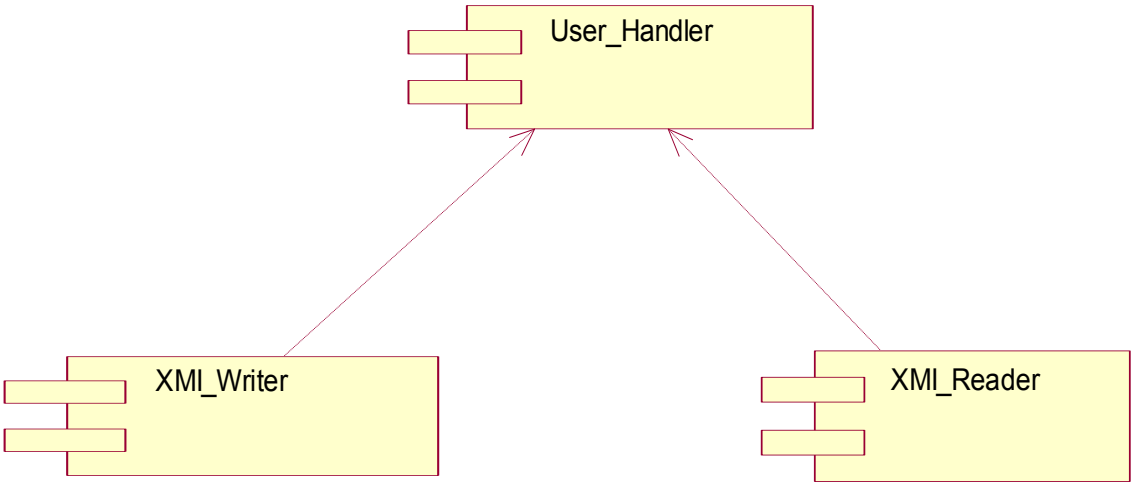


Figure 6.5: User_Handler architecture

The User_Handler component is composed of:

- `XMI_Writer`: Responsible for writing the XMI to document user profiles (as discussed in chapter 4).
- `XMI_Reader`: Responsible for reading information from the XMI representation of the UML user profiles.

6.3 Interaction in the System

Now that we have explained the architecture of our system, we shall move on to describe how the user interacts with the system (Figure 6.6).

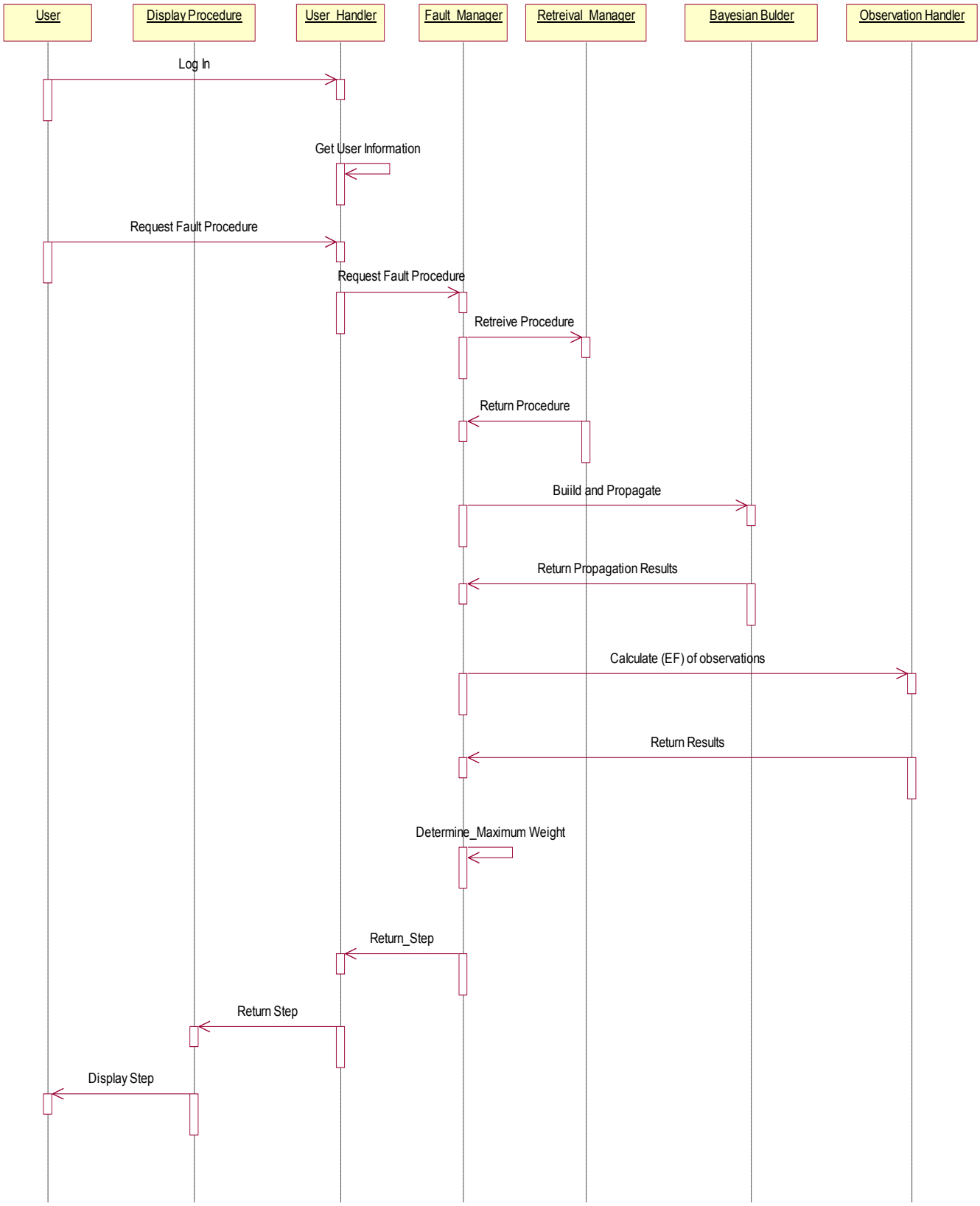


Figure 6.6: Interaction in the system

Interaction with the system goes as follows:

1. The user logs on to the system with a designated username.
2. We get the experience and other information of the user.
3. The user requests a fault procedure.
4. The fault procedure is retrieved from the IETM.
5. A Bayesian Network is constructed.
6. Any evidence is propagated.
7. **Ef** of each observation is calculated.
8. If the cause that the best observation effects, is different from the one that already has the highest weight, the observation to be made by the user is displayed.
9. The cause with the highest weight is displayed to the user.

6.4 Implementation Decisions

The demonstration system implementation is based on the following design decisions:

1. For demonstration purposes and time limitations, not all parts of our methodology are implemented. We will concentrate on the main component of our methodology, which is adaptive fault diagnosis.

2. The Source data for our fault procedure is discussed above and presented in XML format.

3. The system uses an XML-enabled browser to avoid dealing with an extra component to render the XML.

4. Components are implemented as web services (see chapter 2), due to the following advantages web services have over other distributed technologies:
    a. They are an open standard by the W3C [9], and not proprietary to any company.
    b. They communicate with each other and the client via XML (see chapter 2), and since the IETM is written in XML, it is a perfect match up.
    c. They are expandable and highly portable across platforms.

The following sections discuss the tools and the components used in the implementation of the demonstration system.

Platform: Windows XP pro

For ease of development, we chose to run both the client and server on the same machine running the Windows XP operating system

Development Language: C# and .Net technologies

The .Net framework has inherent abilities for XML and Web Services (see chapter 2). That makes it ideal for our development requirements outlined above. It also has an easy to use IDE (Visual Studio.Net) [28] that makes development fast and productive.

Web Server: Internet Information Services (IIS) 5.0

We used IIS 5.0, because it is a robust, proven server that ships with windows XP pro and works perfectly with the .Net framework.

6.4.1 Sample Execution

As depicted in figure 6.6, the first thing the user does is log in to the system. The login screen is presented in figure 6.7. After the user logs on, we retrieve his information and display the available fault diagnosis procedure (figure 6.9). Figure 6.8 displays various measures for demonstration purposes. We see that this user has experience of 4, and at the start all steps have equal weights.

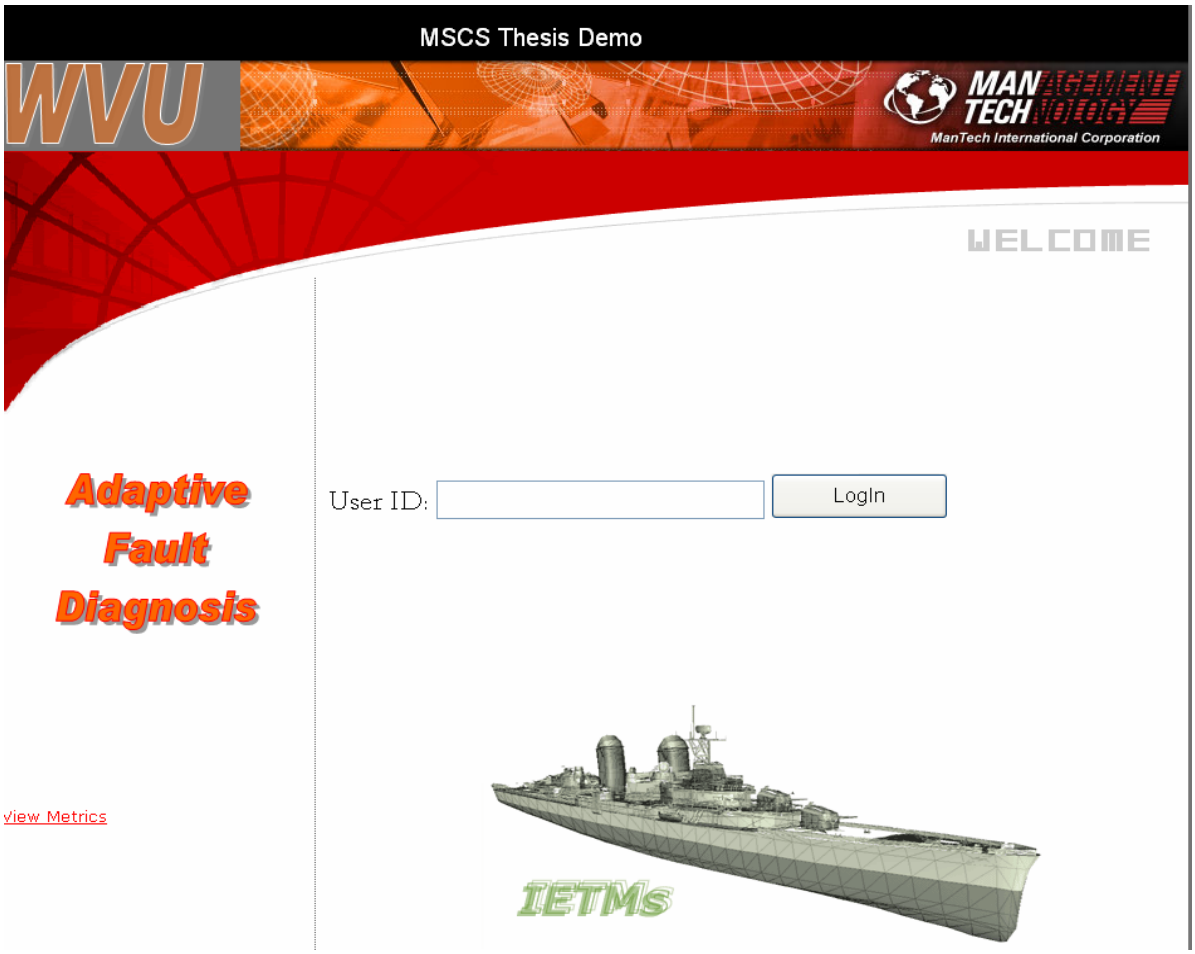


Figure 6.7: System login screen

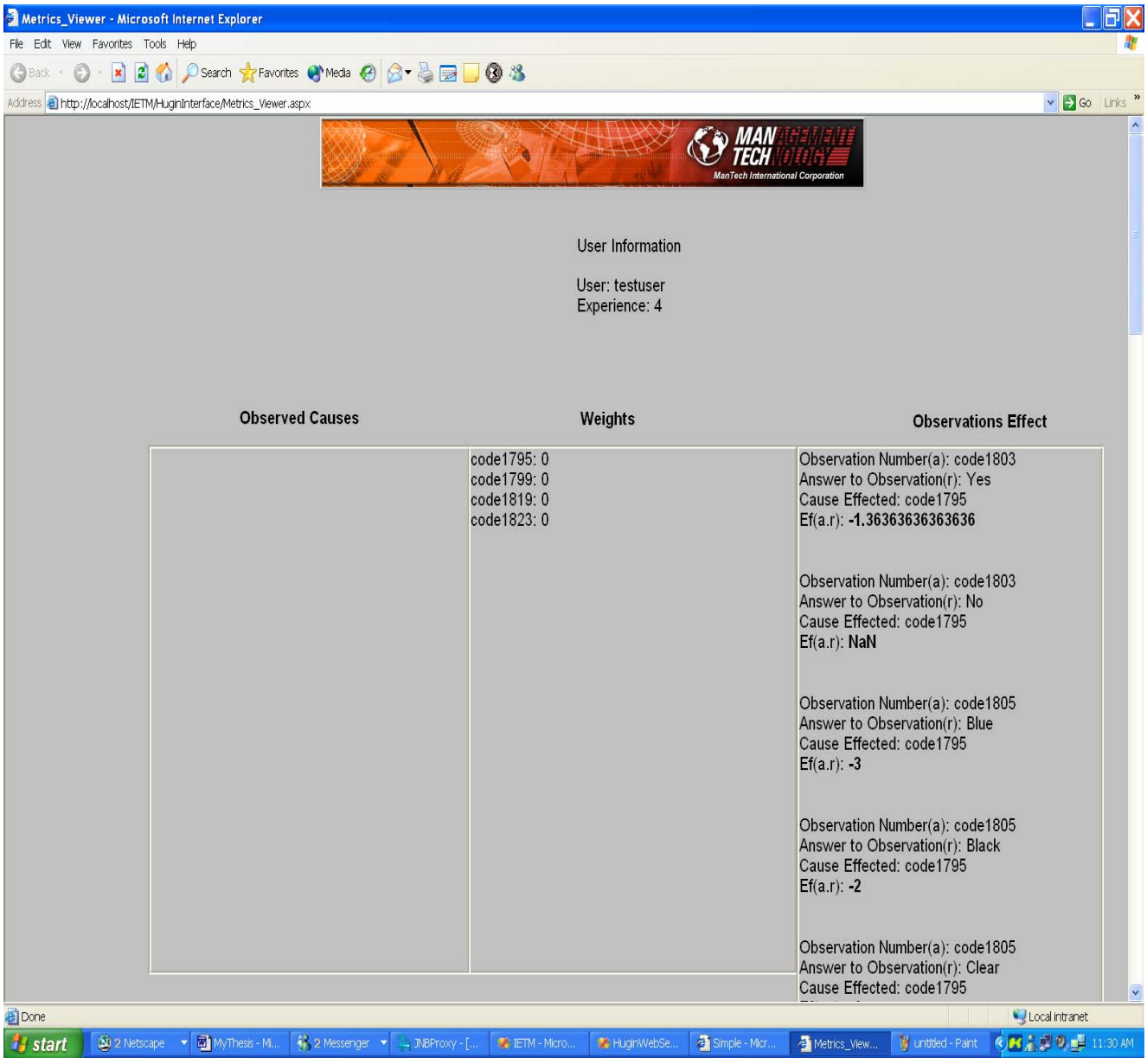


Figure 6.8: The Observer

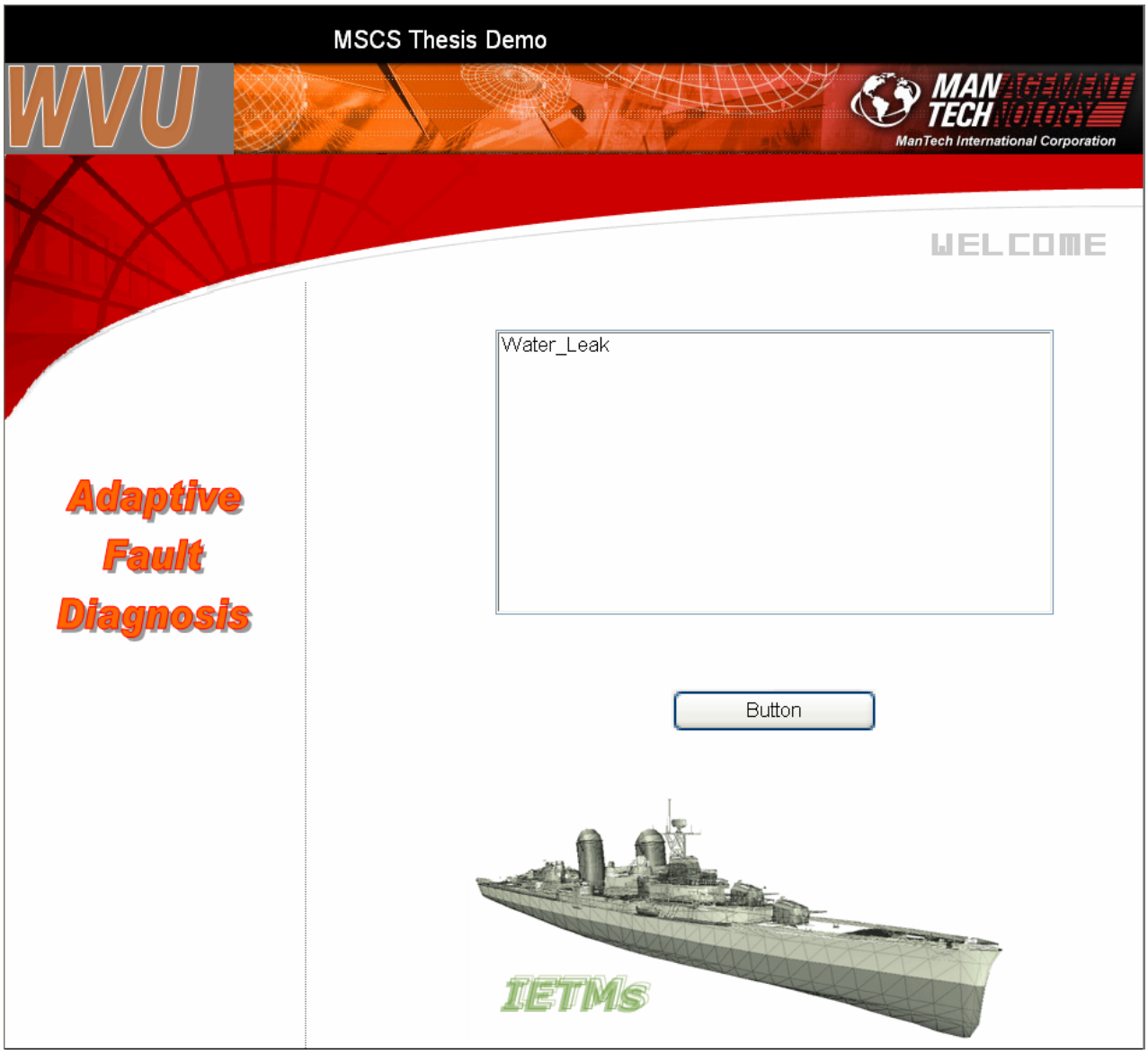


Figure 6.9: Listing of available fault diagnosis modules

After the user selects the "Water_Leak" procedure, we see that the cause: code1799 which is "Faulty Washer Pump" (figure 6.1) has the highest weight. Although, as we can see from table 6.1, this doesn't have the highest probability, but it has an appropriate cost (0.3), and the one with the highest probability ("Faulty Radiator Pump") doesn't have the highest weight because it has the a cost 6, hence it might be difficult for the user to do (user's experience is 4).

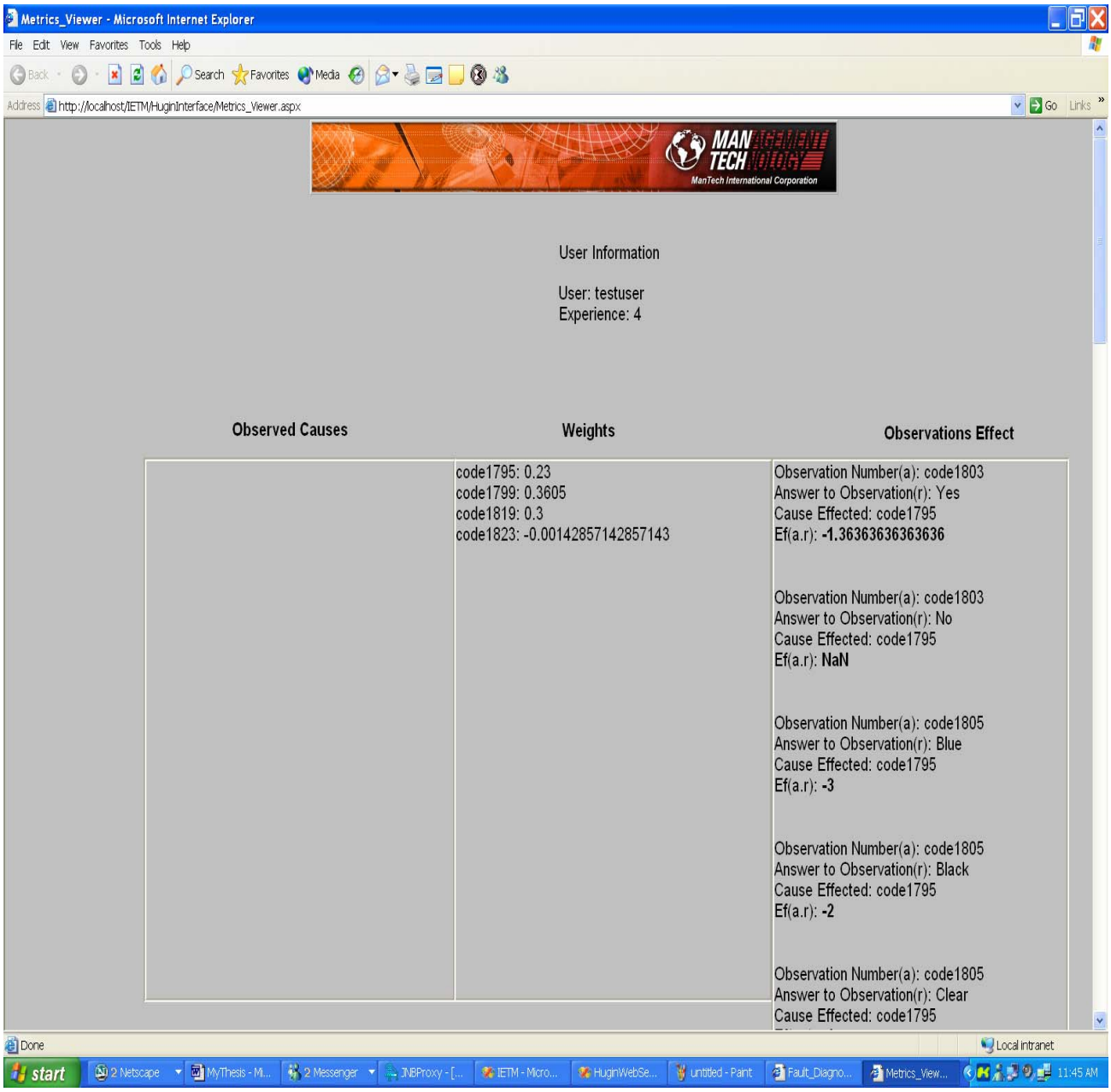


Figure 6.10: Propagation Results after a user with E =4 requested procedure

However, if a user with experience 9 logs in (figure 6.11), "Fault Radiator Pump", which has the highest probability will have the highest weight, because its cost (6) is lower than this user's experience, then the user can perform this step with ease.

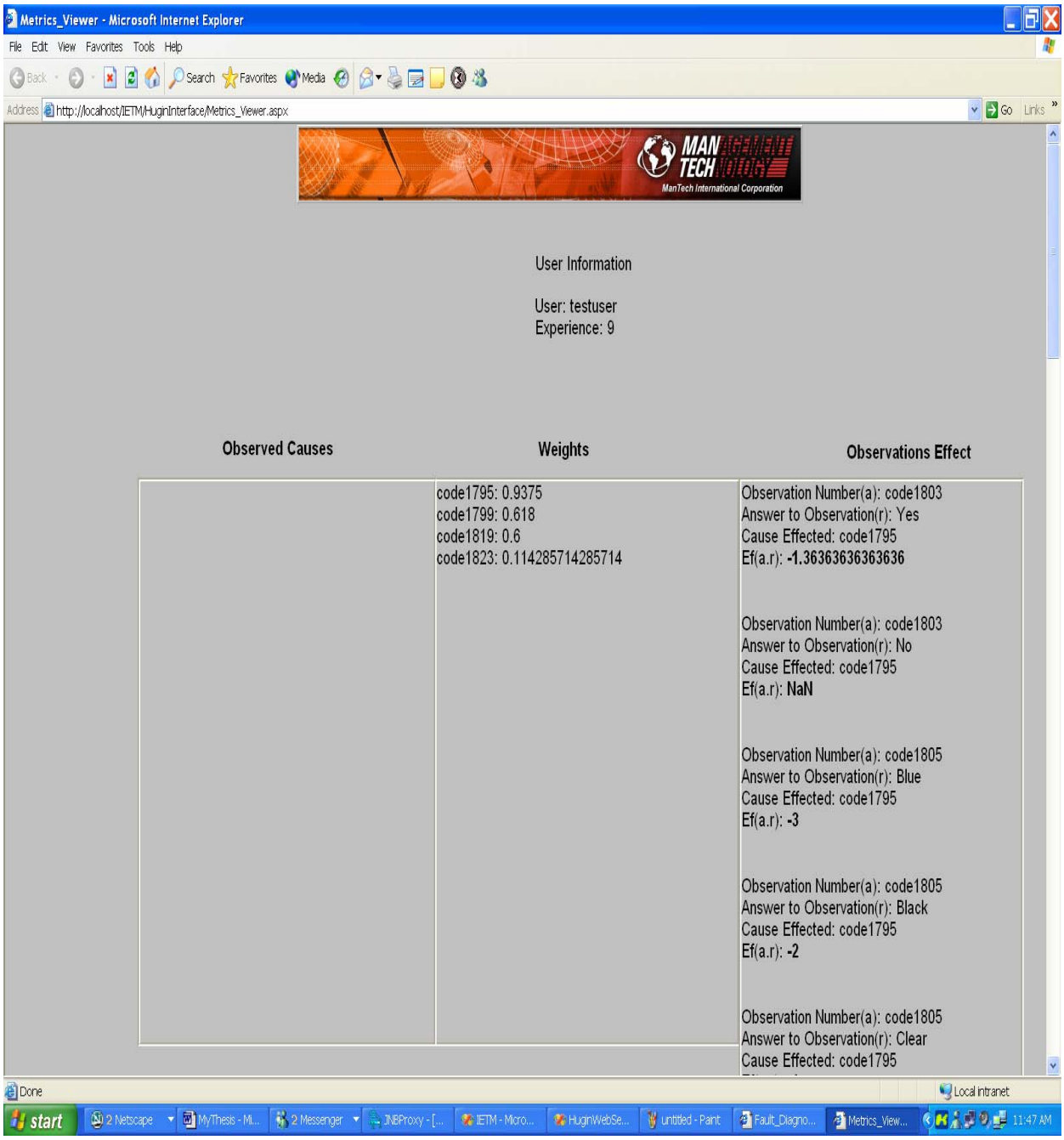


Figure 6.11: Propagation Results after a user with E =9 requested procedure

Moreover if a user with E=1 (the lowest allowable experience level) logs in (figure 6.13), then "Faulty Oil Pump" will have the highest weight, since its cost is 1, hence it is the only step the user can perform with ease. The logic behind that is that it doesn't make sense to

present a user a step that he can't perform, although it may have a higher probability of fixing the fault.

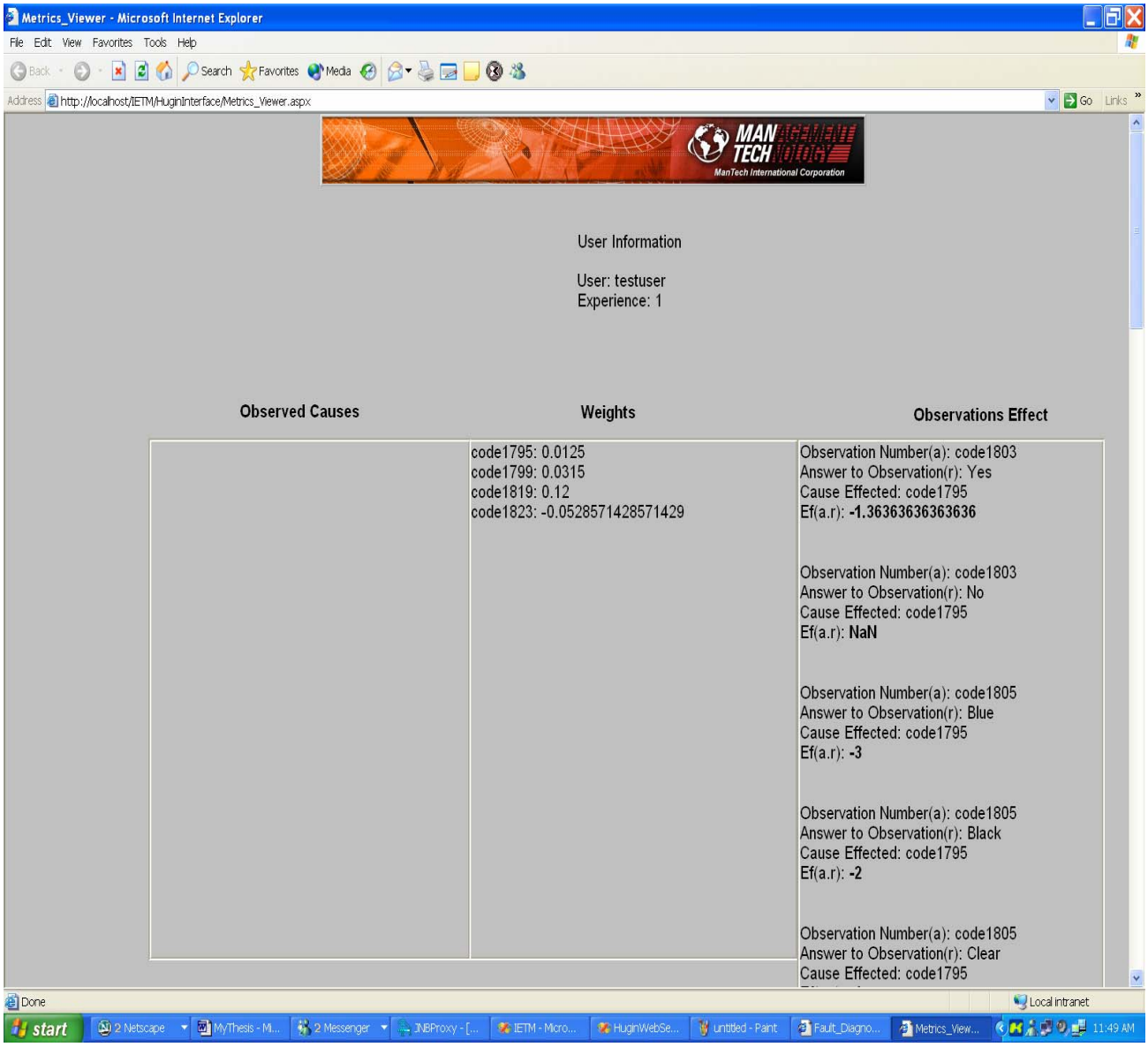


Figure 6.13: Propagation Results after a user with E =1 requested procedure

It is pertinent to say that no questions or observations were presented to the user, because no observation effected any cause (**Ef**<0), hence the user was spared being asked any questions.

So we can see that our system adapts to each particular user, presenting diagnostic steps that best fit his abilities.

CHAPTER 7: CONCLUSION AND FUTURE WORK

7.1 Implementation Issues

The choice of using XML web services to implement our component based architecture was a good choice; it provides an extensible framework which can be extended if added functionality is needed in the future. It was also a good choice because communication between web service components is done via SOAP messages, using XML, which makes them ideal for our cause.

The choice of the .Net framework to build web services was also a good choice, because this framework is built from the ground up based on XML and web services technology. It also provides us an easy to use IDE, Visual Studio .Net, which dramatically cuts development time, and contains extensive libraries for handling XML and SOAP.

The use of Rational Rose for our modeling effort was essential to out approach. Rational Rose is the leading tool for modeling software artifacts using UML, so it was a natural choice.

7.2 Conclusion

Current IETMs do not contain the necessary adaptation methodologies that enable them to be truly adaptive to the with the user. In this thesis we presented a methodology for adaptation in IETMs. We focused on developing a framework for adaptive fault diagnosis in IETMs. Our Approach is summarized in the following:

- Develop an IETM object model to better understand the structure of our IETM
- Model the user dynamically using UML as he/she navigates through the system, in order to use this information to adapt the IETM to his/her specific characteristics

- Use Bayesian networks to represent the diagnostic steps of a fault procedures
- Use Data mining techniques to recognize patterns between answers to questions that users are asked and ultimate causes of a particular fault, hence diminishing the number of questions a user is asked and only asking questions when they effect the final outcome of a fault procedure

We also developed a demonstration system (chapter 6) that proved that our methodology works and the system was adapted to the user that was using it.

7.3 Future Work

As we have stated before, we do not claim that we have developed a full fledged class V IETM; we only concentrated on developing a framework for adaptation in fault diagnosis in IETMs. Hence in the future, we would like to do the following:

1. Develop an adaptation framework for descriptive as well as procedural information in IETMs.
2. Develop a specific methodology for tracking user behavior in IETMs, using the OO user profiles discussed above.
3. Port our user tracking system to other non IETM related systems
4. Develop a full fledged class V IETM
5. Analyze the performance of our proposed web services based architecture.

One of the major tasks we would like to see is to develop a full fledged Class V IETM. This means the system should exhibit intelligent behavior and should be based on relational databases rather than XML files, although these data bases might be structured in XML. We would like to develop an AI based systems that provides the functionality of Class V IETM. One approach is to use Neural Network based techniques to provide the "learning" functionality and to be able to predict future user inputs and adapt the IETM accordingly. Another approach will be to develop an Expert System that can achieve

similar functionality. It is important to point out any of the above mentioned approaches needs large amounts of data that the systems must be trained on, or in case of an expert system, rules that can be incorporated into the system.

We would also like to develop an authoring system that incorporates adaptation data from the time the IETM is originally authored, hence providing full IETM life cycle support for adaptation in IETMs.

BIBLIOGRAPHY

1. Naval Surface Warfare Center, Caderock Division. *Data Base, Revisable – Interactive Electronic Technical Manuals for the support of MIL-PRF-87269A.* <http://navysgml.dt.navy.mil/ietm/ietm>.
2. U.S Department of Defense. <http://www.dod.gov>.
3. The European Association of Aerospace Industries. <http://www.aecma.org>.
4. AECMA specification S1000D. <http://www.tpsmg.org>.
5. Dina Ghobashy. (2001). *Web – enabled supported alternative to Interactive Electronic Technical Manual Database (IETMDB) Specification.*
6. International Organization for Standardization. Information Processing – Text and Office systems – Standard Generalized Markup Language (SGML). ISP 8879:1986. <http://www.oasis-open.org/cover/general.html>.
7. The Unified Modeling Language. <http://www.uml.org>.
8. The Extensible Meta Language. <http://www.xml.org>.
9. The World Wide Web Consortium. <http://www.w3.com>.
10. The Microsoft Corporation Web Services portal. <http://www.microsoft.com/webservices>.
11. Booch, G, Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1999.
12. Dan Livingston.: Advanced SOAP for Web Development. Princeton Hall, 2002.
13. Microsoft Corporation .Net Framework website. <http://www.microsoft.com.net>.
14. Finn V. Jensen. : Bayesian Networks and Decision Graphs. Springer-Verlag, 2001.
15. Stupido Ghosal, etal (1999). An Integrated Process for System Maintenance, Fault Diagnosis and Support. *Proceedings of the IEEE Aerospace Conference, Aspen, Colorado.*

16. Claus Skaanning, Finn V. Jensen and Uffe Kjaerulff (1999). SACSO – A Bayesian Network Tool for Automated Diagnosis of Printing Systems.

17. Finn V. Jensen, Uffe Kjaerulff and Brian Kristainsen (2000). The SACSO Methodology for Troubleshooting Complex Systems.

18. Claus Skaanning (2000). A Knowledge Acquisition Tool for Bayesian-Network Troubleshooters.

19. Eric Horvitz (1997). Lumiere Project: Bayesian Reasoning for Automated Assistance.

20. International Standards Association. <http://www.iso.org>.

21. David Carlson. Modeling XML Applications with UML: Practical Business Applications. Addison Wesley, 2001.

22. Peter Brusilovsky (1996). Methods and techniques of adaptive hypermedia.

23. The Object Management Group. <http://www.omg.org>.

24. Timothy J. Grose, Gary C. Doney, Stephen A. Brodsky. Mastering XMI: Java Programming with XMI, XML and UML. John Wiley and Sons, 2002.

25. Morris H. DeGroot, Mark J. Schervish. Probability and Statistics. Addison-Wesley, 2002.

26. Hugin Expert® Tool. <http://www.hugin.com>.

27. T. Menzies and Y. Hu (2002). Just Enough Learning (of association rules). *Published in the WVU CSEE Tech Report.*

28. Visual Studio.Net. <http://msdn.microsoft.com/vstudio>.

29. Kalagnanam, J. & Henrion M (1990). A comparison of decision analysis and expert rules for sequential analysis in P. *Uncertainty un Artificial Intelligence 4*, North-Holland, New York, pp 271-281.

30. Kjaerulff U. & van der Gaag, L. C. (2000). Marking sensitivity analysis computability efficient, Submitted to UAI 2000.

31. Deb, S., Pattipati, K.R. and Shrestha, R (1997). QSI's Integrated Toolset. *Published in IEEE Autotestcon, Anaheim, CA, pp. 408-421.*

32. Deb, S., Ghosal, S., Marthur, A., Shretha, R and Pattipati, K.R (1998). Multisignal Modeling for Diagnosis FMECA and Reliability. *Published in IEEE SMC.*

33. Pattipati, K.R. etal(1994). TEAMS: Testability Engineering and Maintenance System". *Published in IEEE ACC, Baltimore, MD.*

34. Hackermanm D. etal. (1995). Decision-theoretic troubleshooting. *Published in the communication of the ACM, pp 49-57.*

35. Alfred Kobsa, etal. Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships.

36. Ardissono, L. and Foy, A. (1999). Tailoring the Interaction with Users in Electronic Shops. *Published in the proceedings of the seventh international conference on user modeling.*

37. Balabanovic, M. (1997). An Adaptive Web Page Recommendation Service. *Published in the Proceedings of the 1st International Conference on Autonomous Agents.*

38. Wolfgang Pohl, Ingo Schwab and Ivan Koychev. Learning About the User: A General Approach and Its Application.

39. Gerhard Weber and Marcus Speecht. User Modeling and Adaptive Navigation Support in WWW-based Tutoring Systems.

40. Alexander Kuenzer, Christopher Schlick, etal. An Emperical Study of Dynamic Bayesian Networks for user Modeling.

41. Multipurpose Internet Mail Extensions <http://www.mhonarc.org/~ehood/MIME/MIME.html>.

Appendix A S1000D Object Model

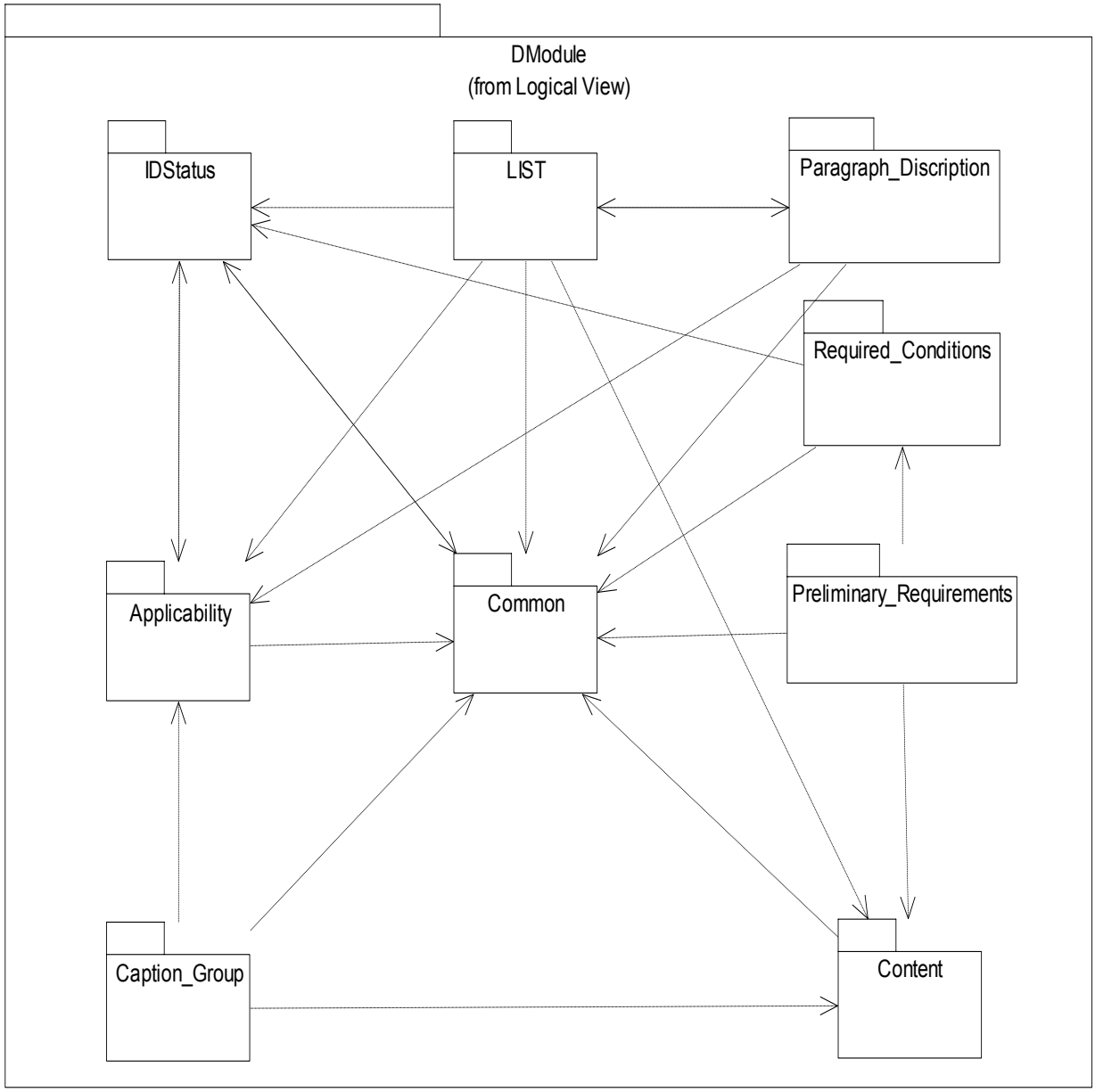


Figure A.1: Top Level Package Hierarchy

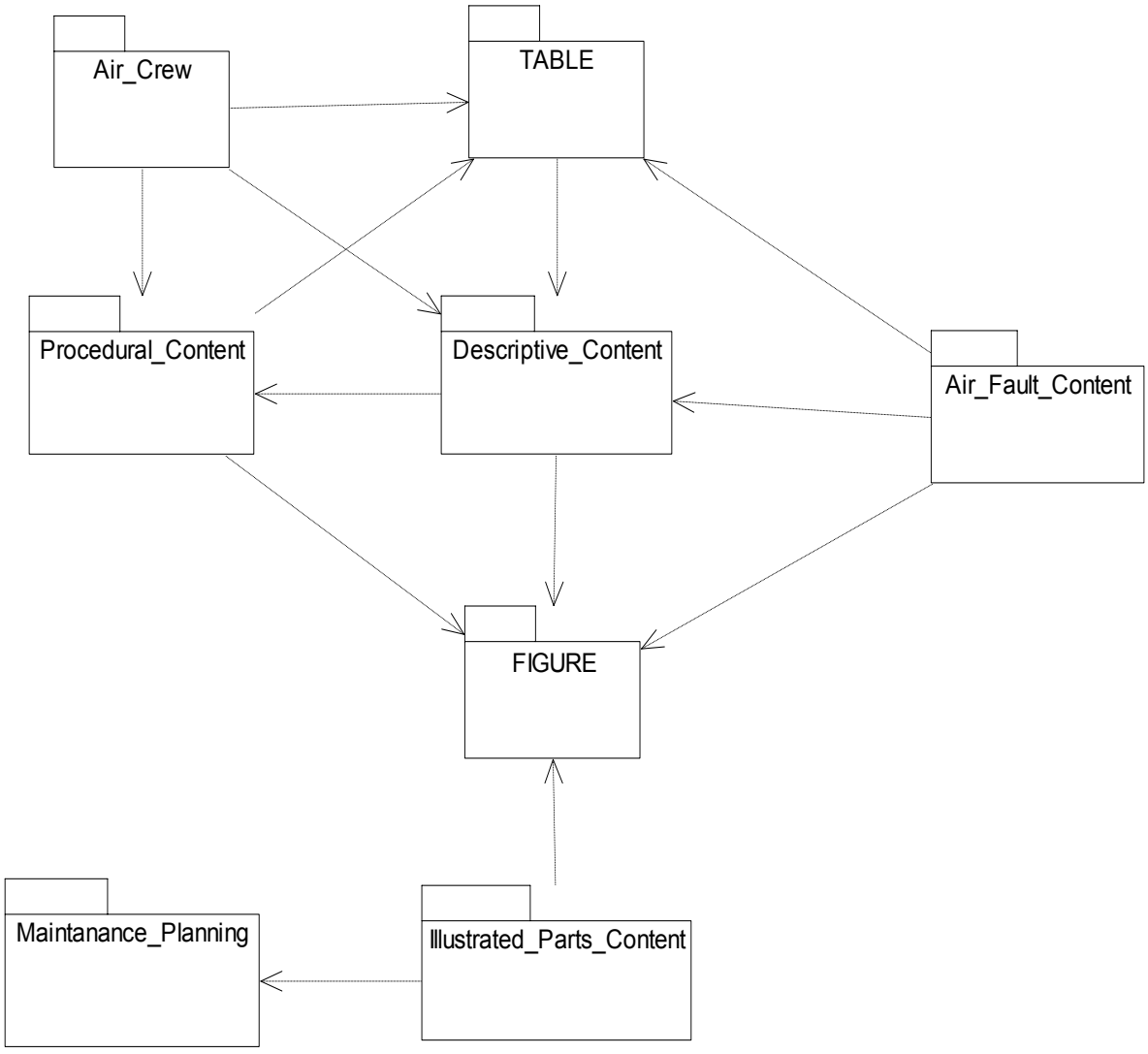


Figure A.2: Content Package Diagram

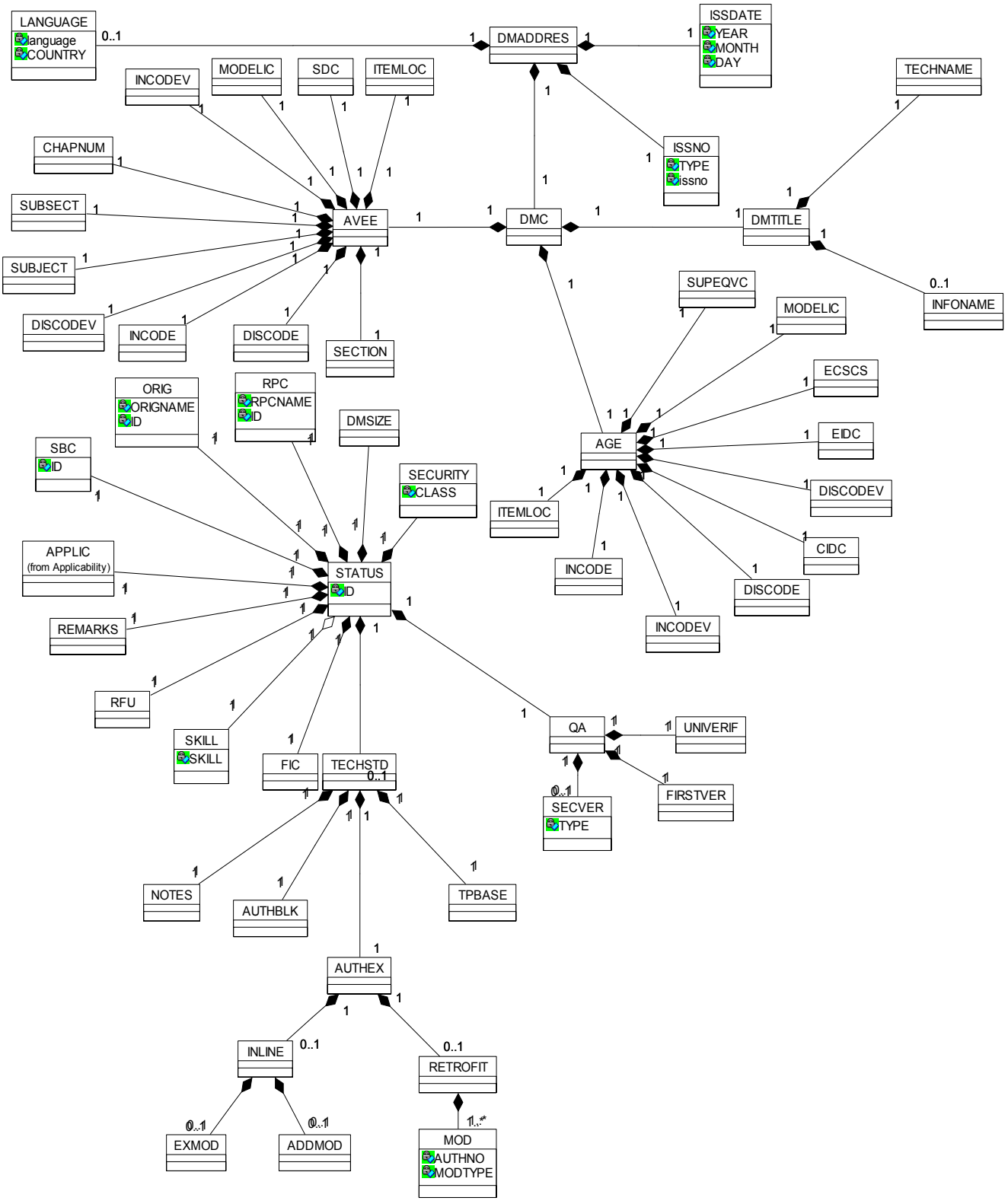


Figure A.3: IDSTATUS Package

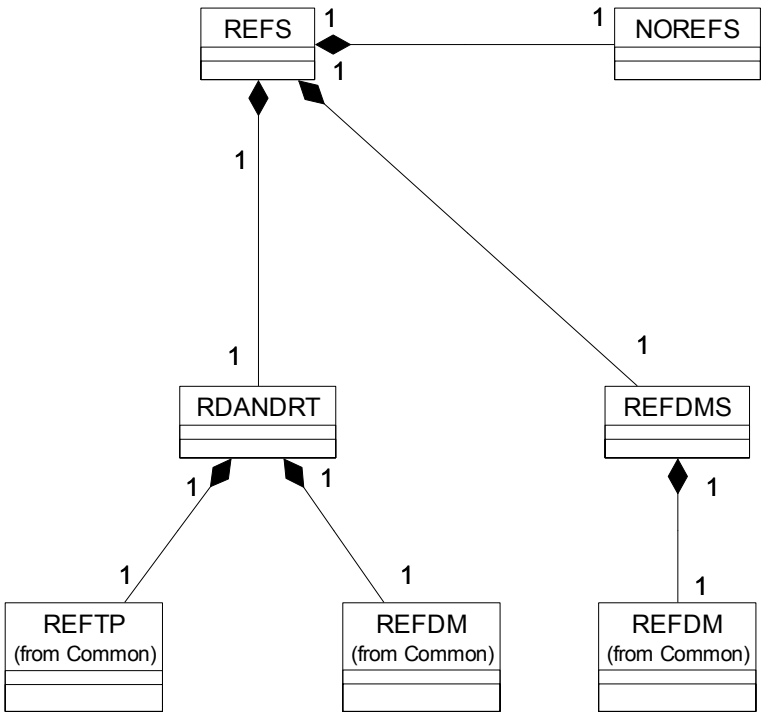


Figure A.4: Reference Elements

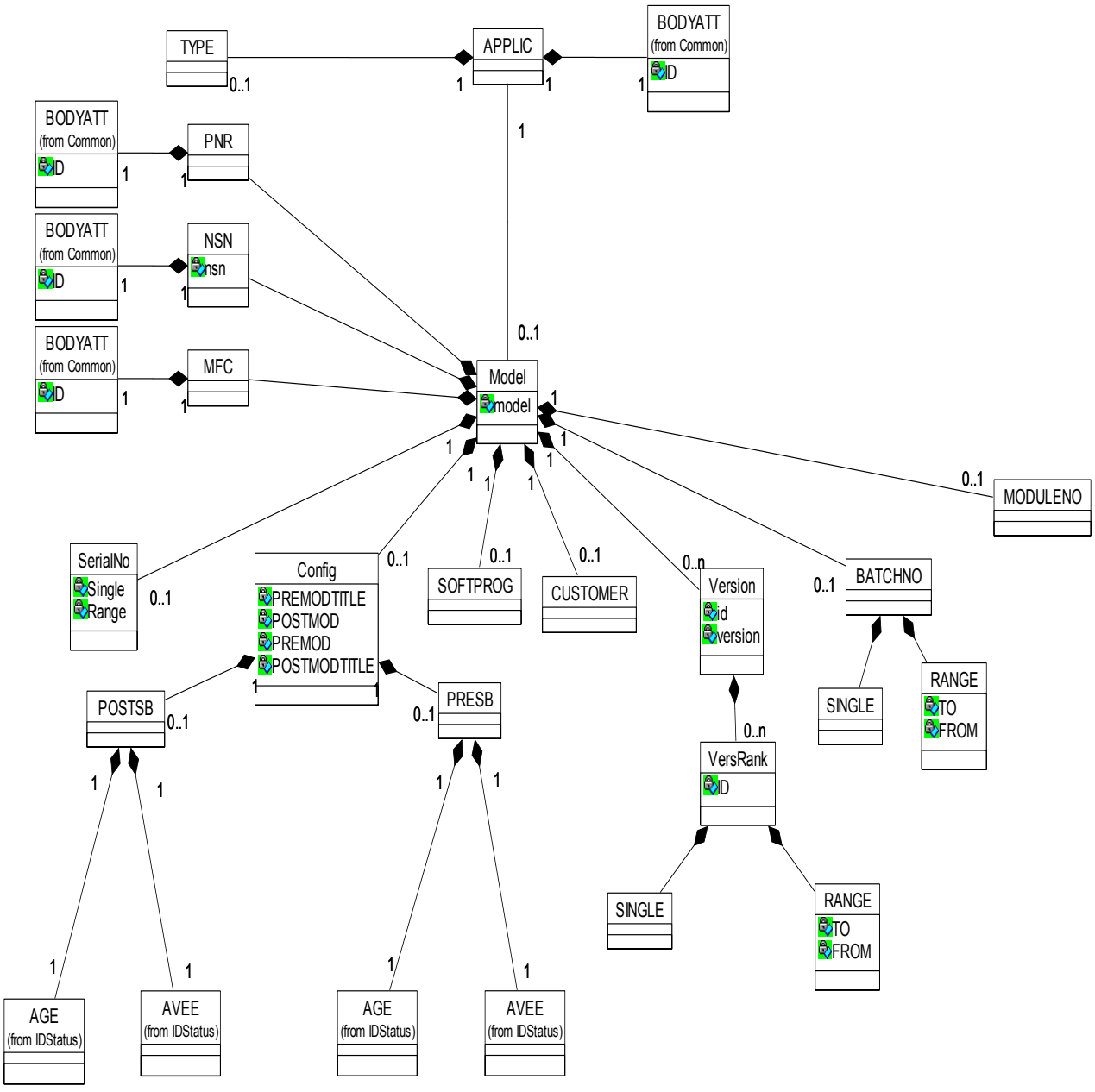


Figure A.5: Applicability Package

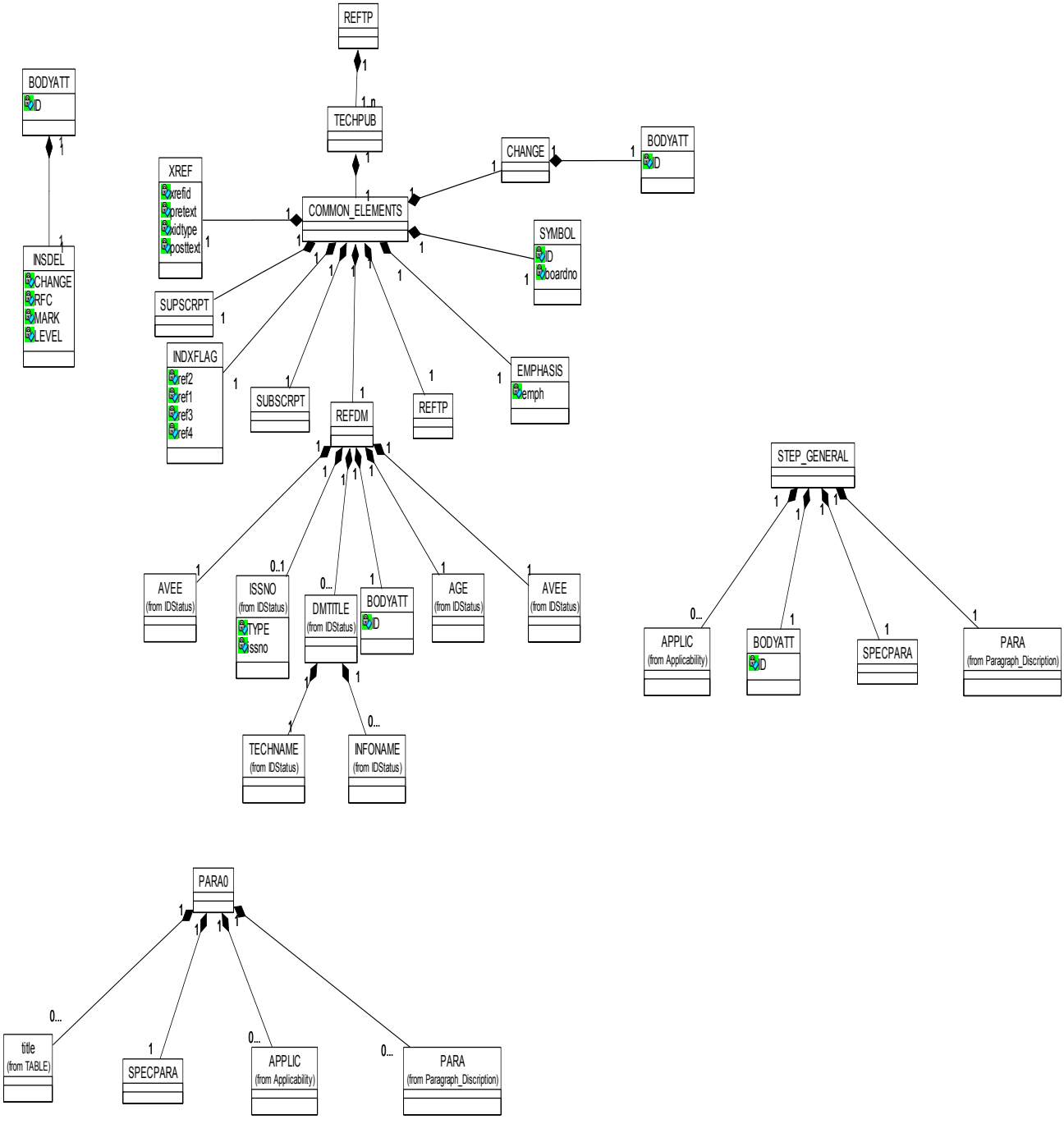


Figure A.6: Common Package. Set of Common Elements between all packages

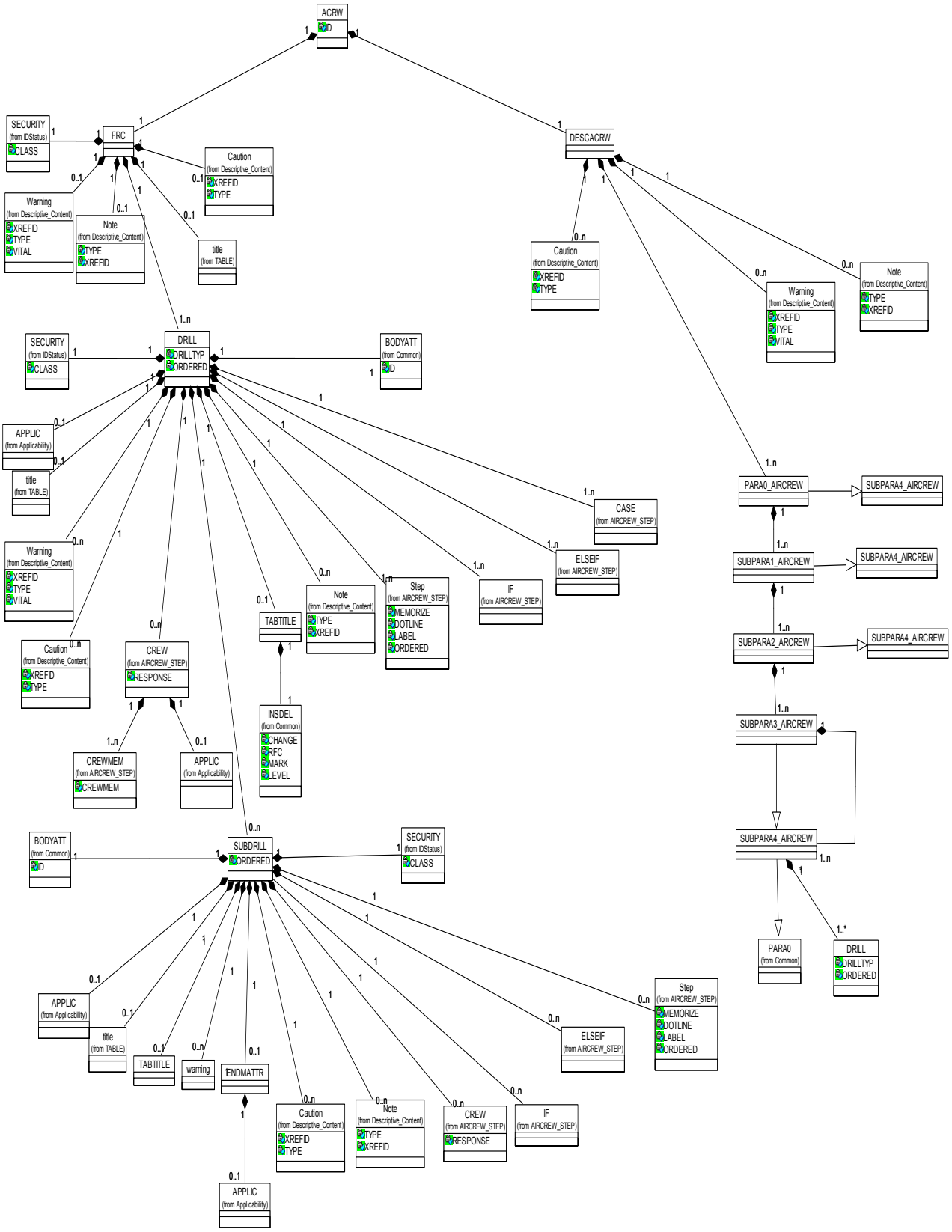


Figure A.7: Aircrew Package

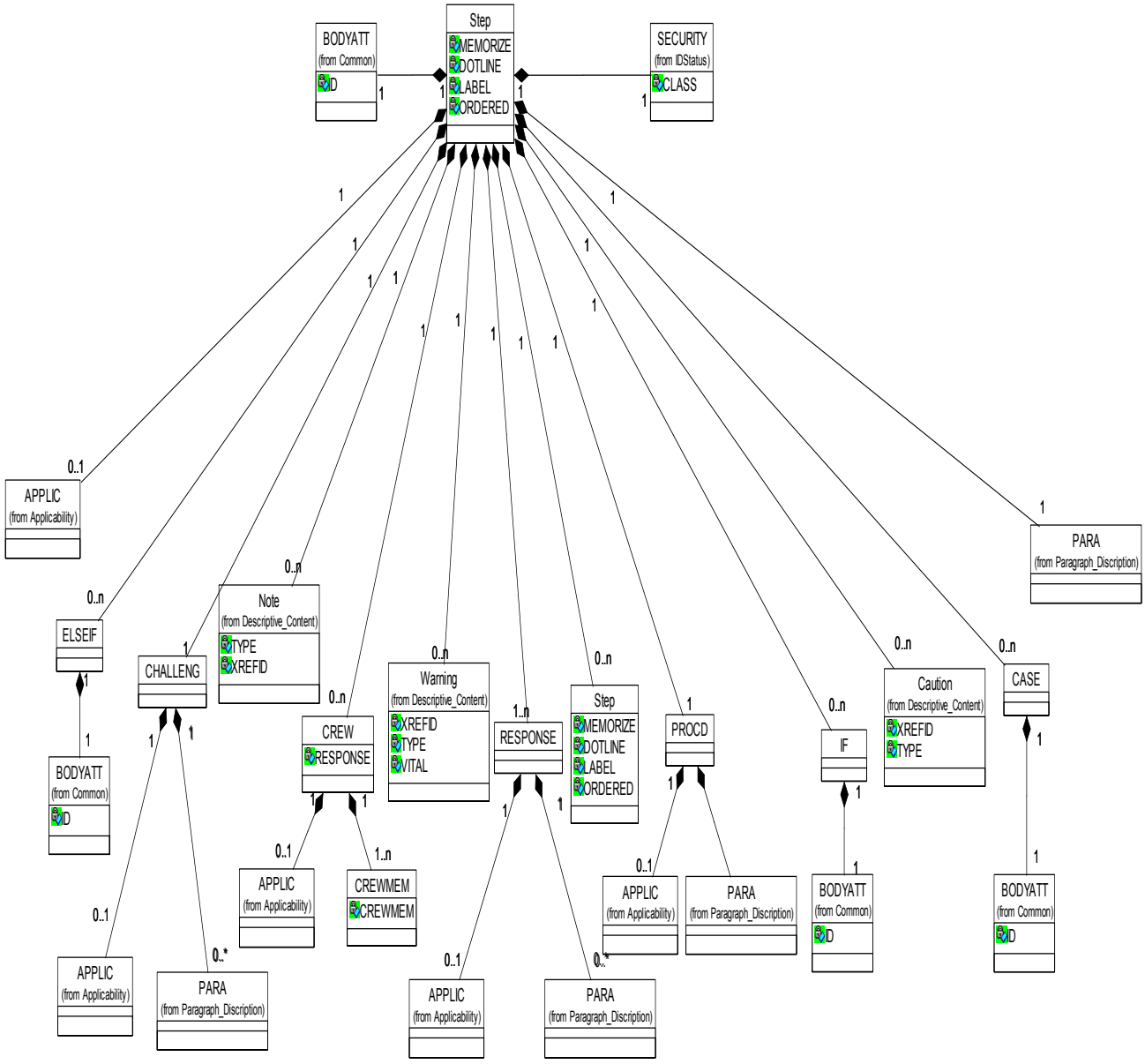


Figure A.8: Aircrew step elements

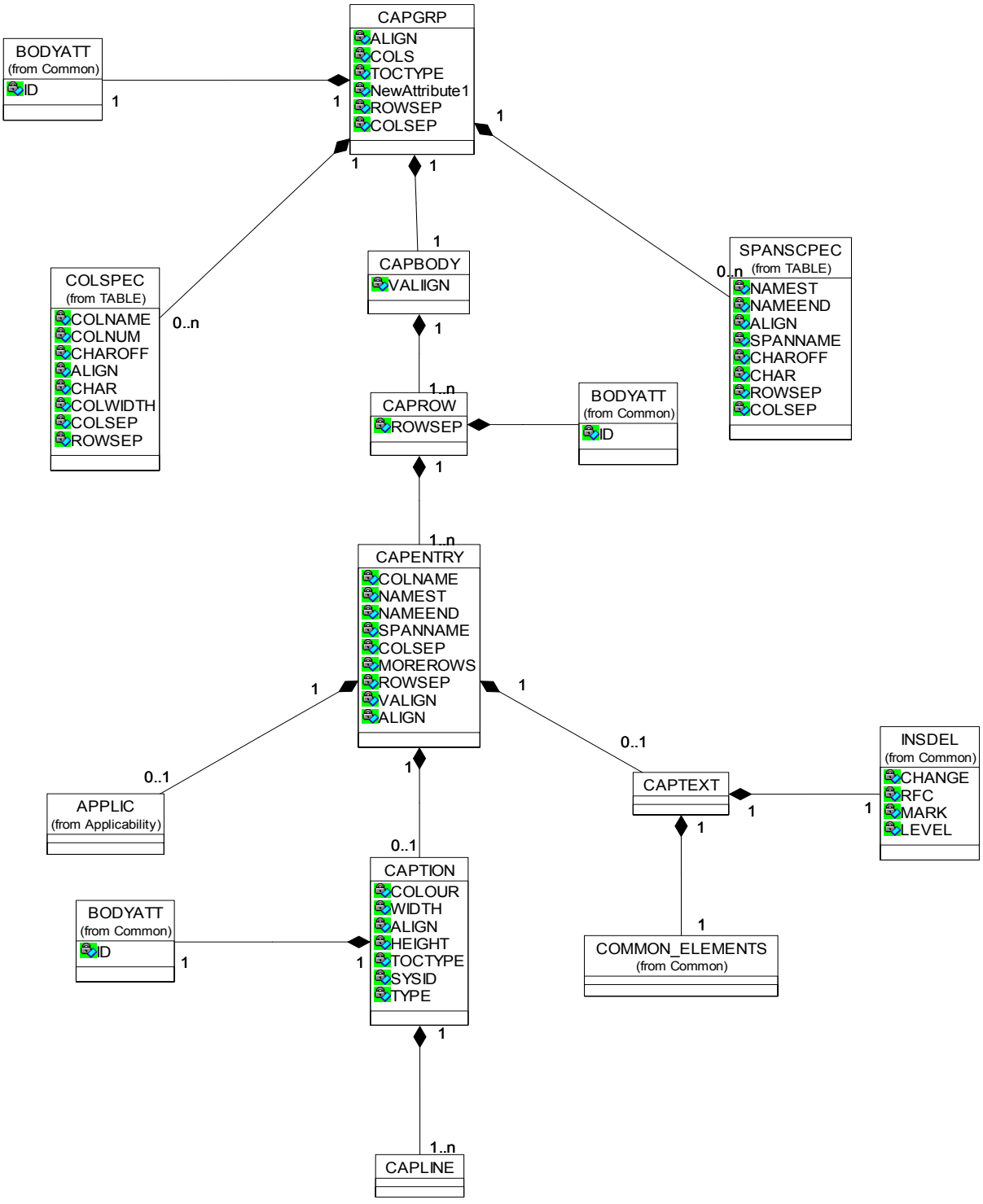


Figure A.9: Caption Group Elements

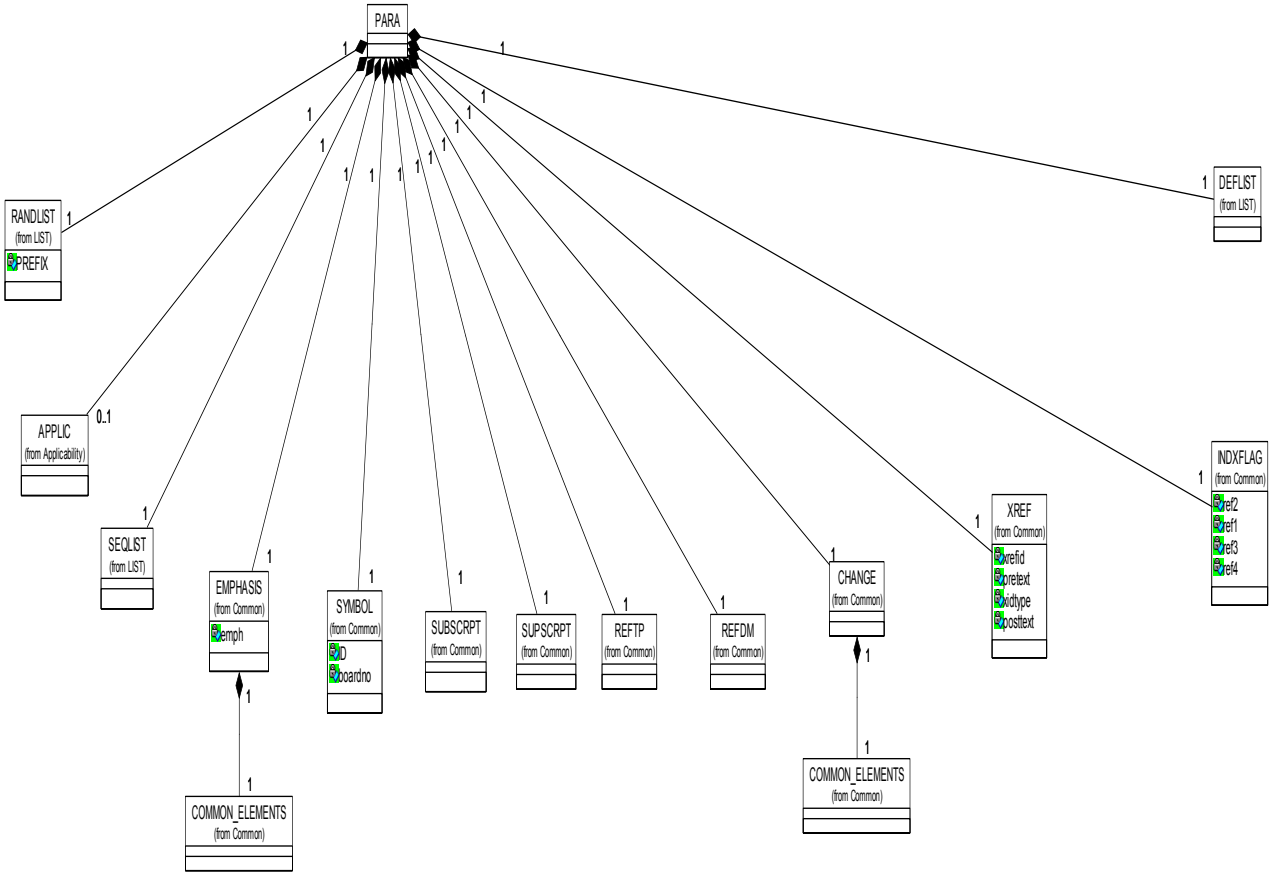


Figure A.10: Paragraph Description Elements

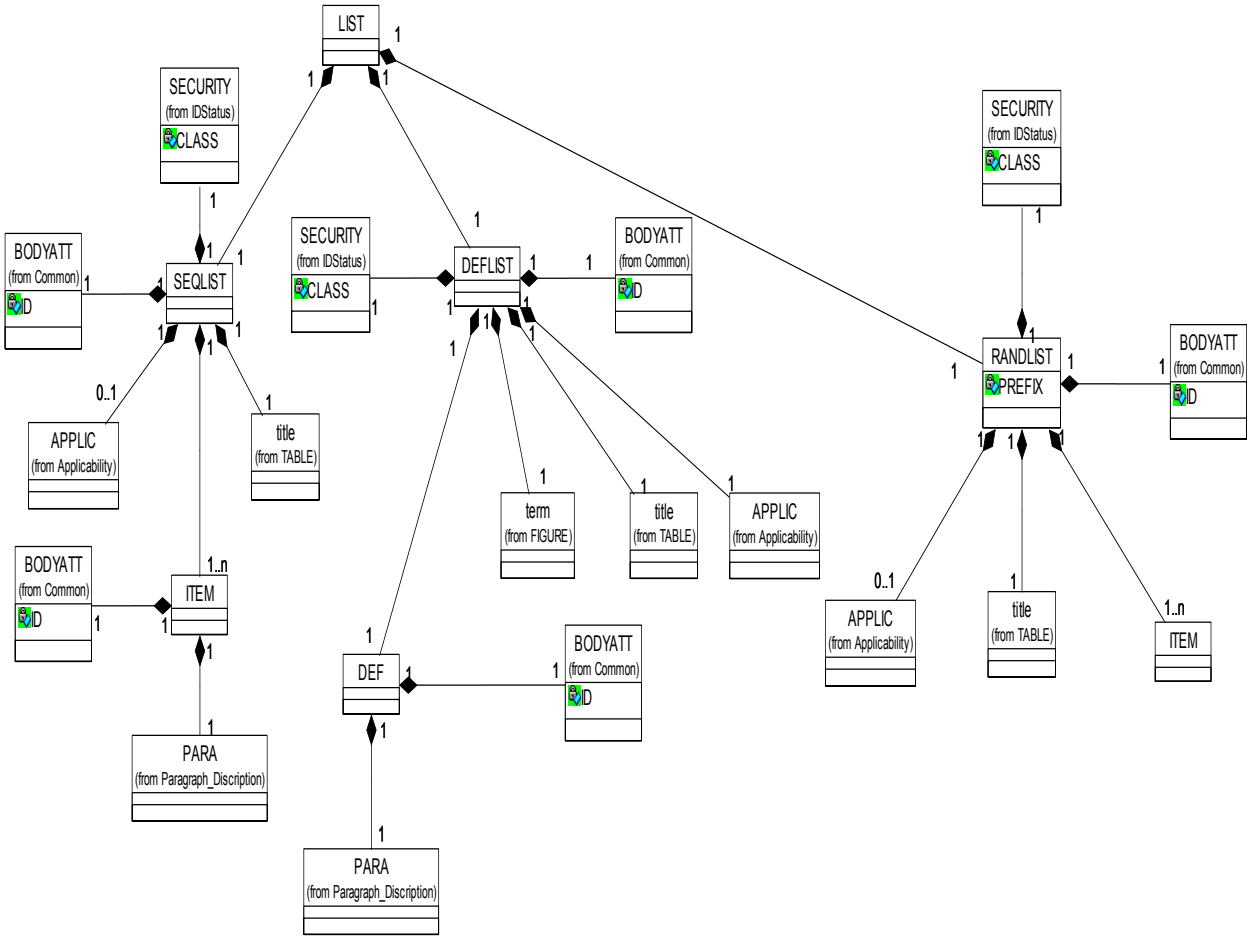


Figure A.11: List Description Elements

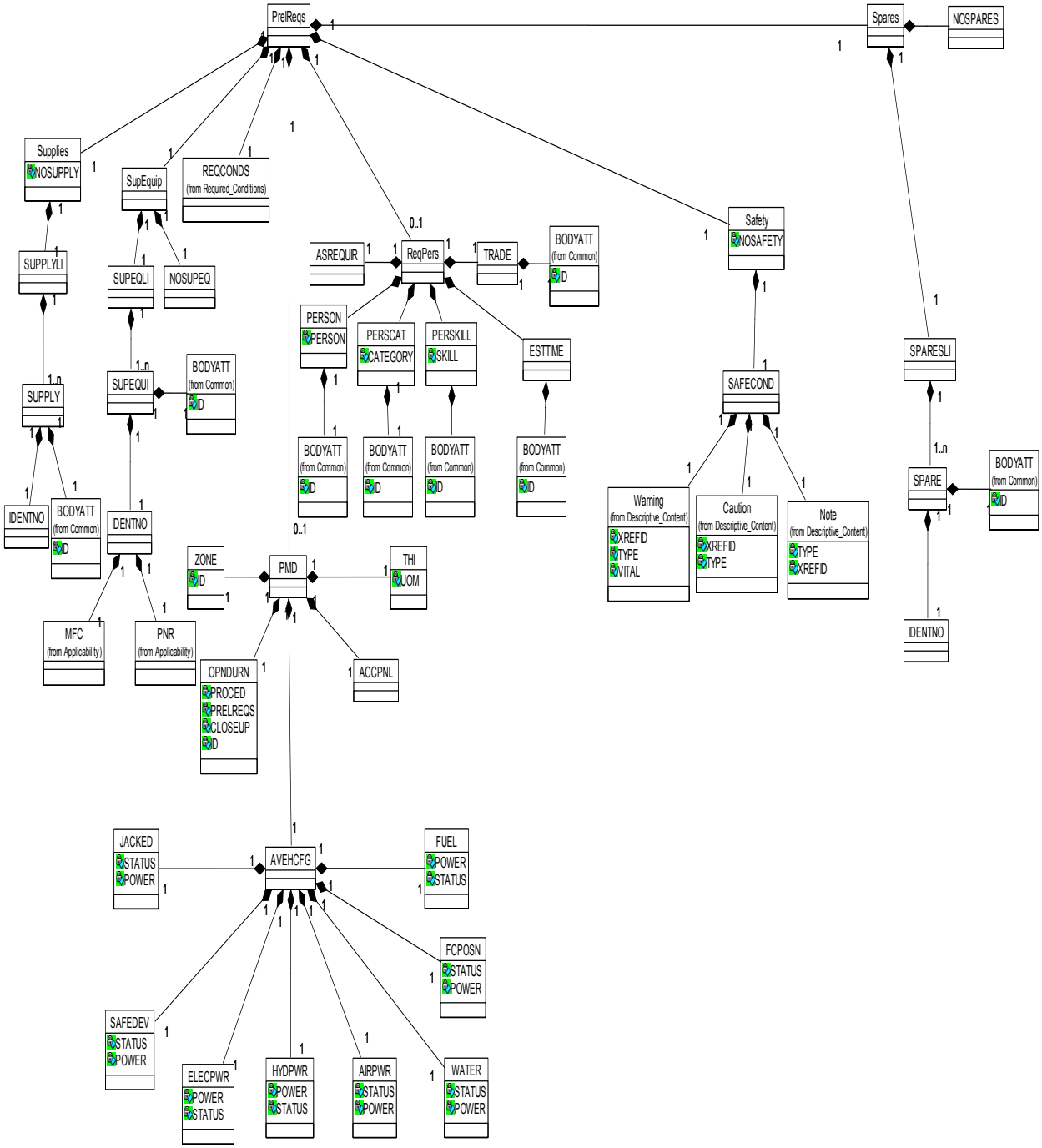


Figure A.12: Preliminary Requirements Elements

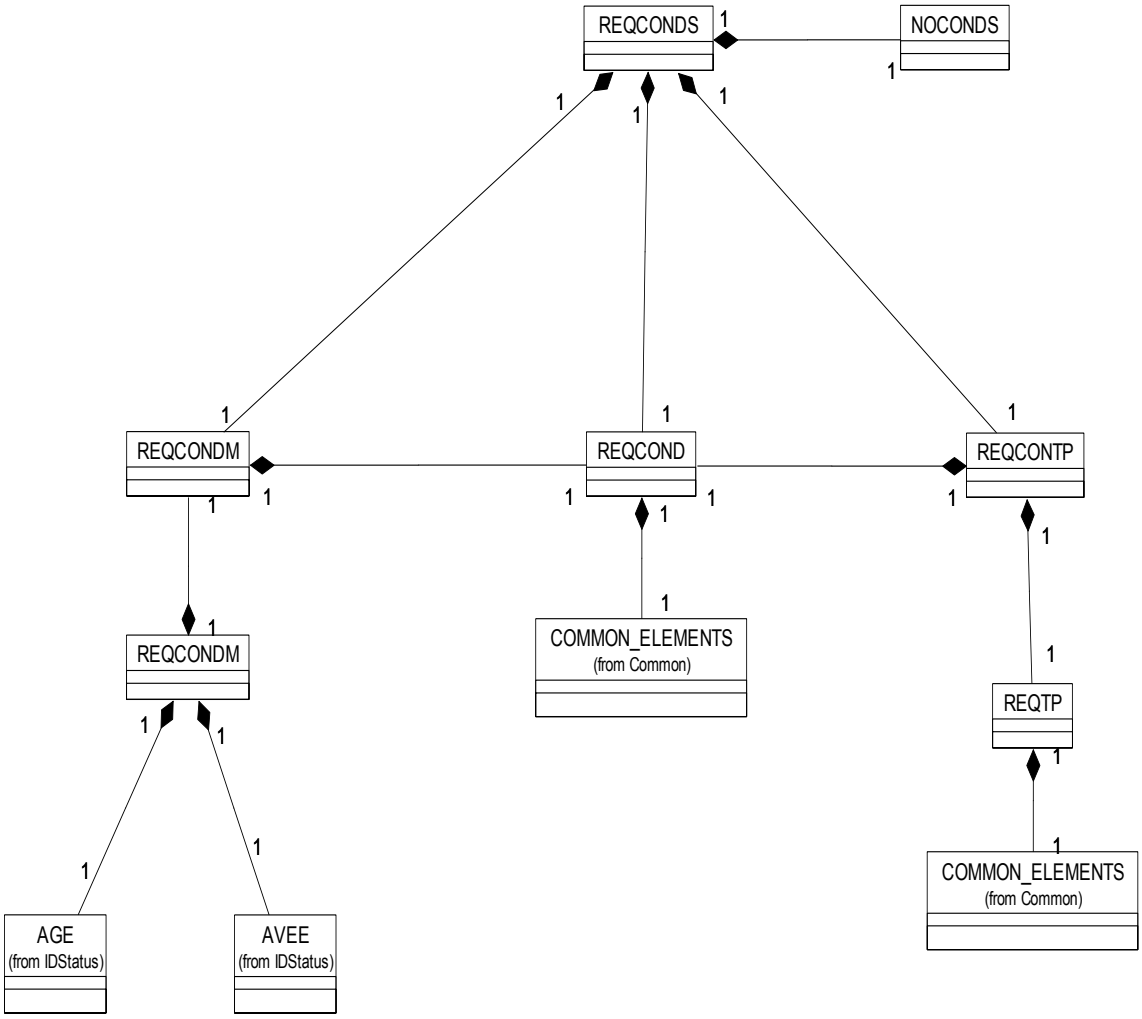


Figure A.13: Required Conditions Elements

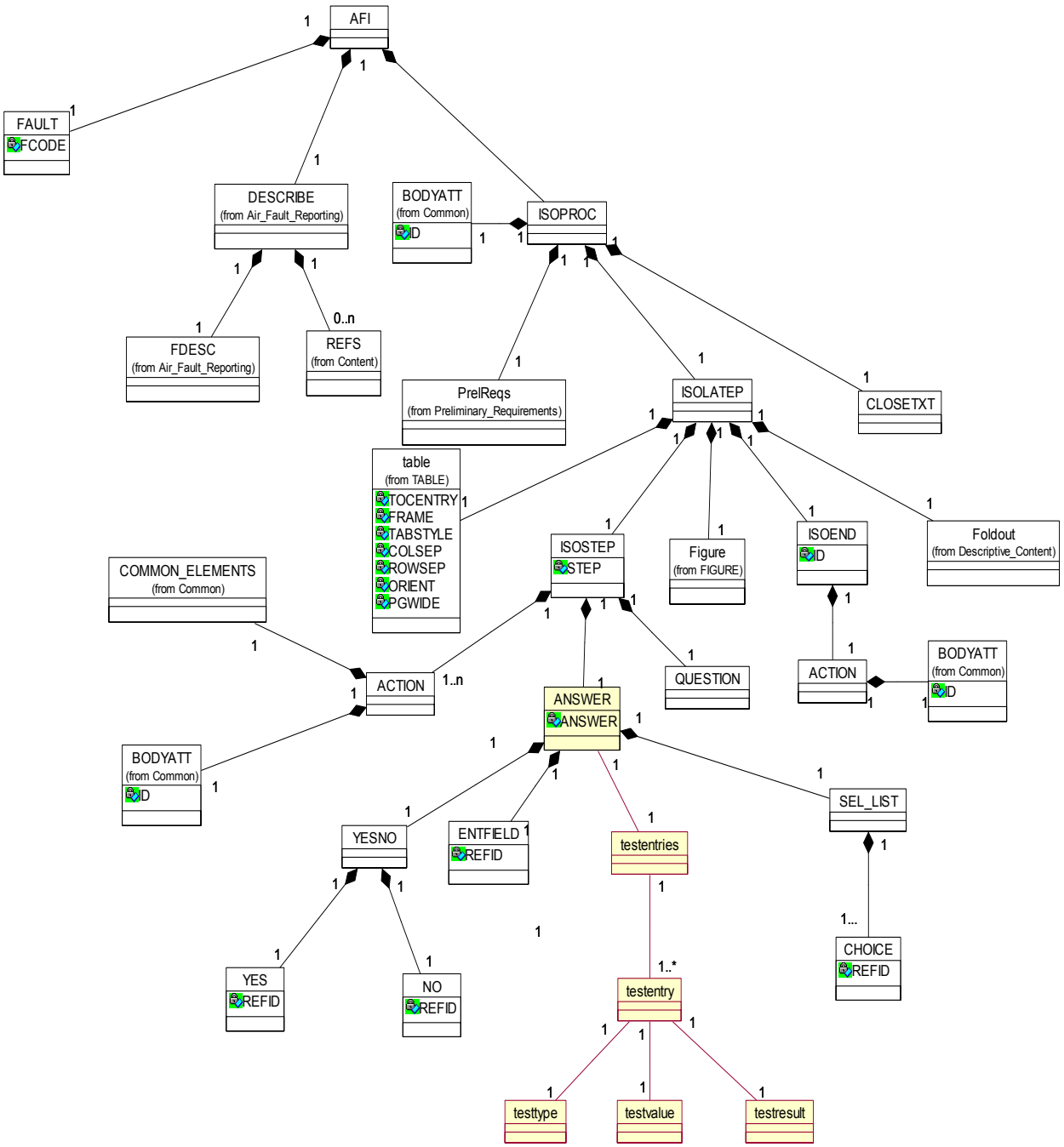


Figure A.14: Air Fault Isolation Elements

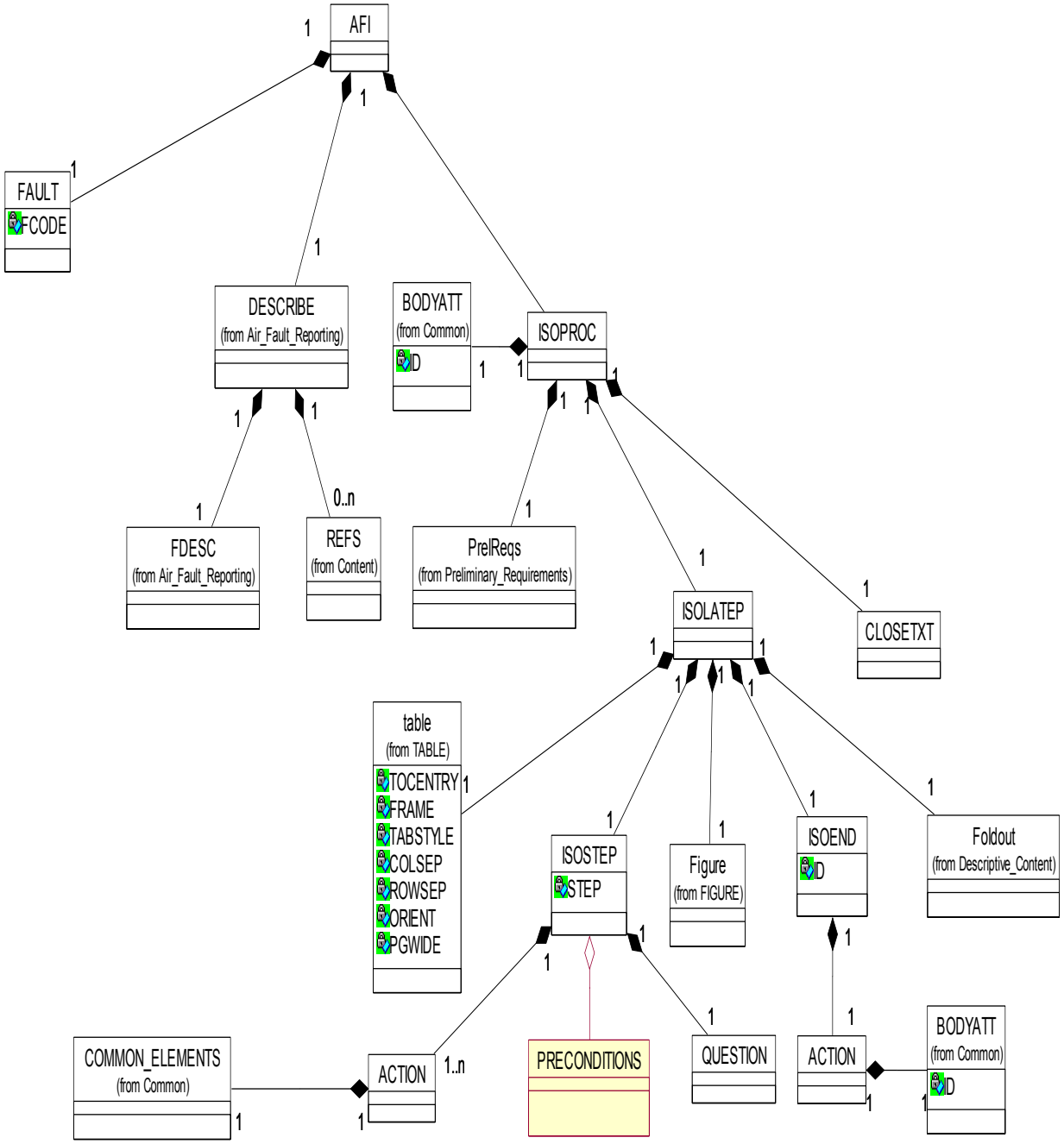


Figure A.15: Air Fault Isolation Elements with preconditions

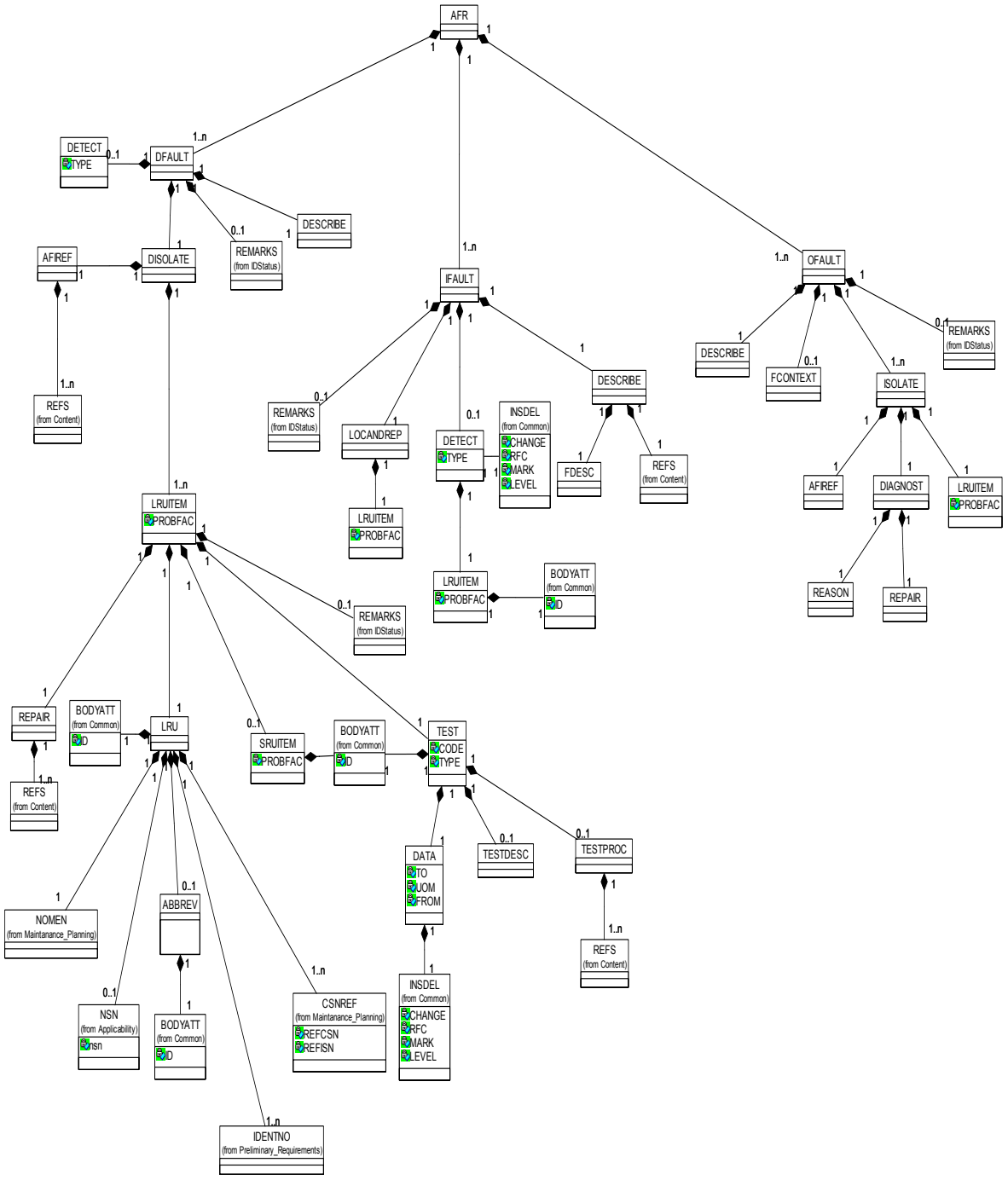


Figure A.16: Air Fault Reporting Elements

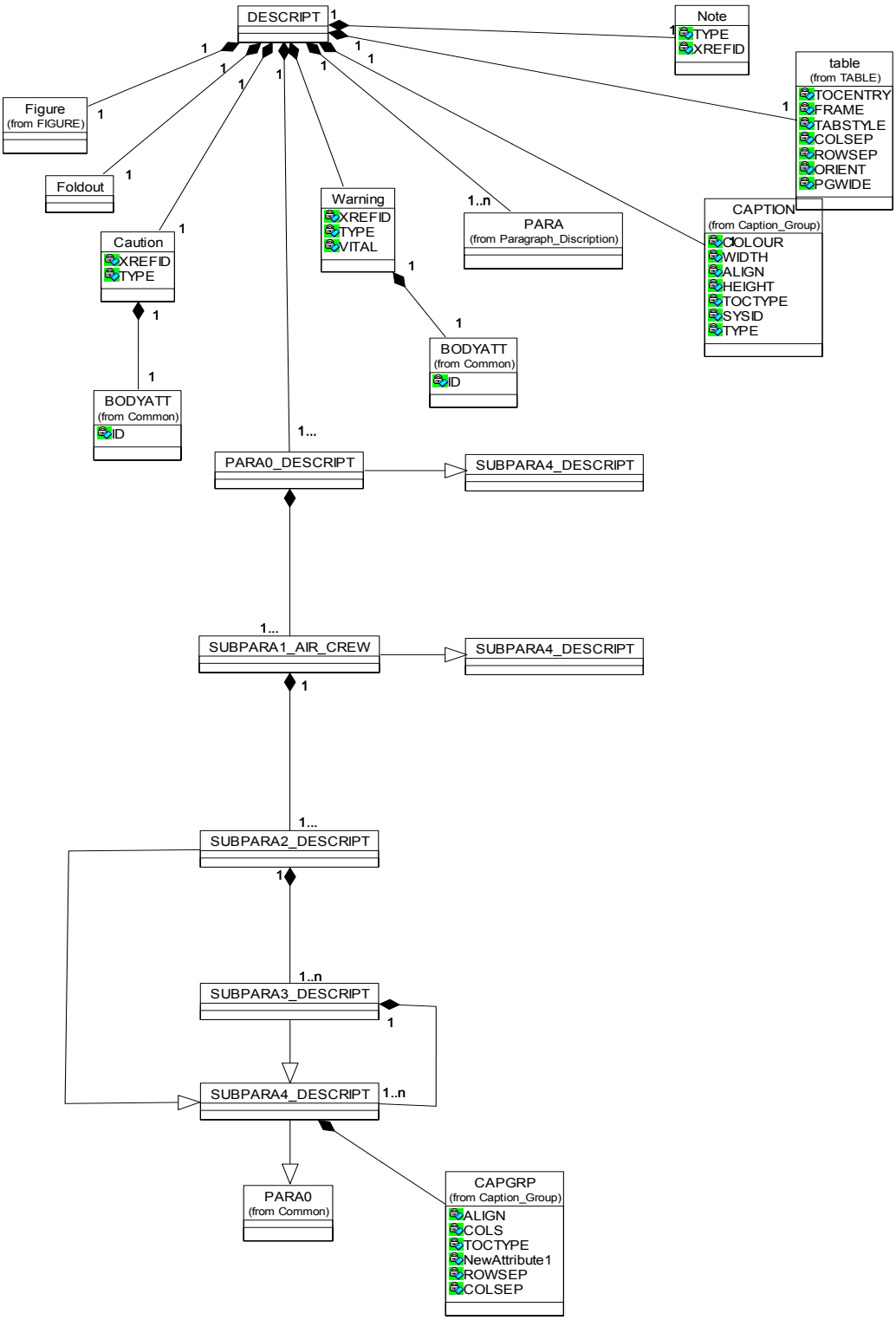


Figure A.17: Descriptive Information Elements

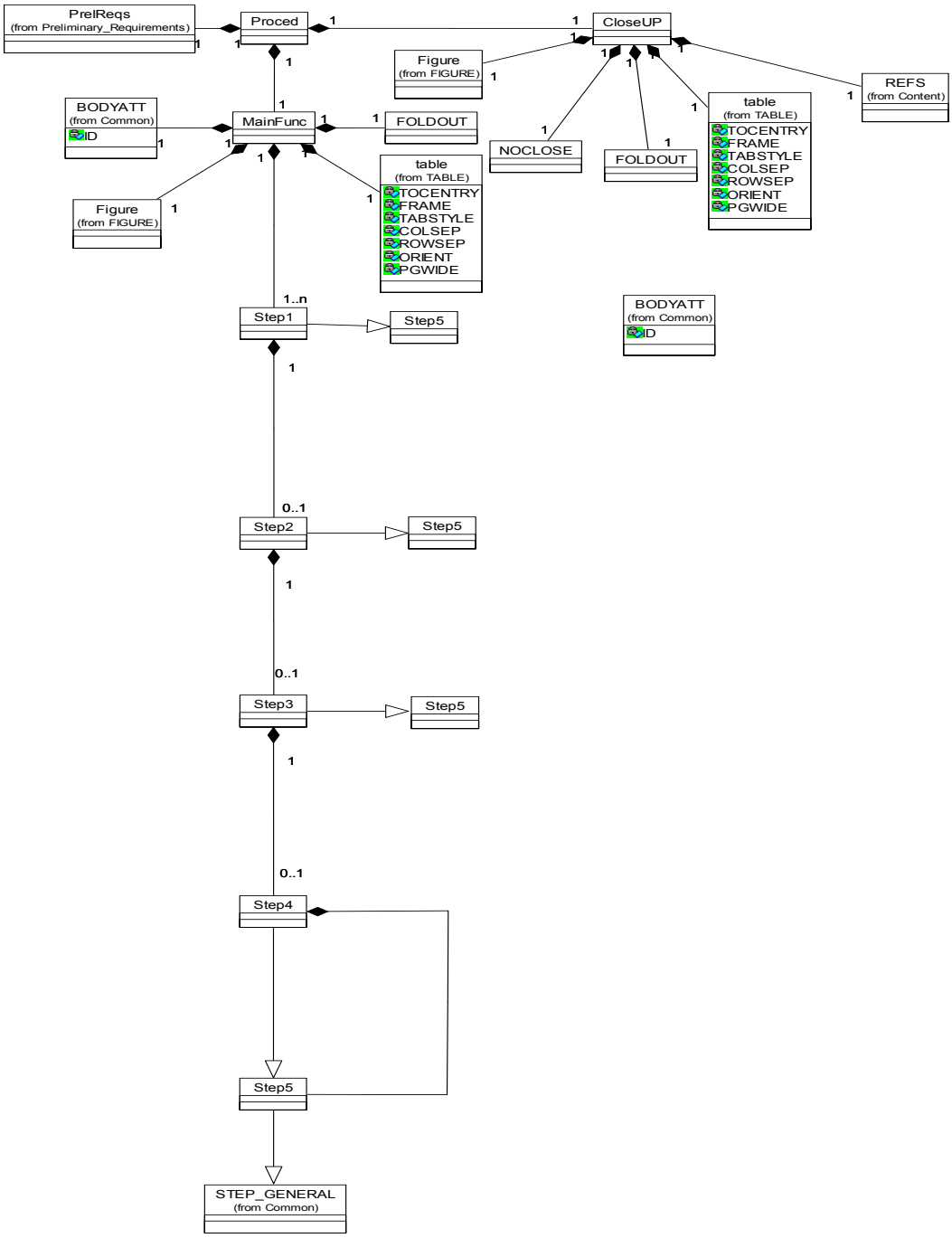


Figure A.18: Procedural Information Elements

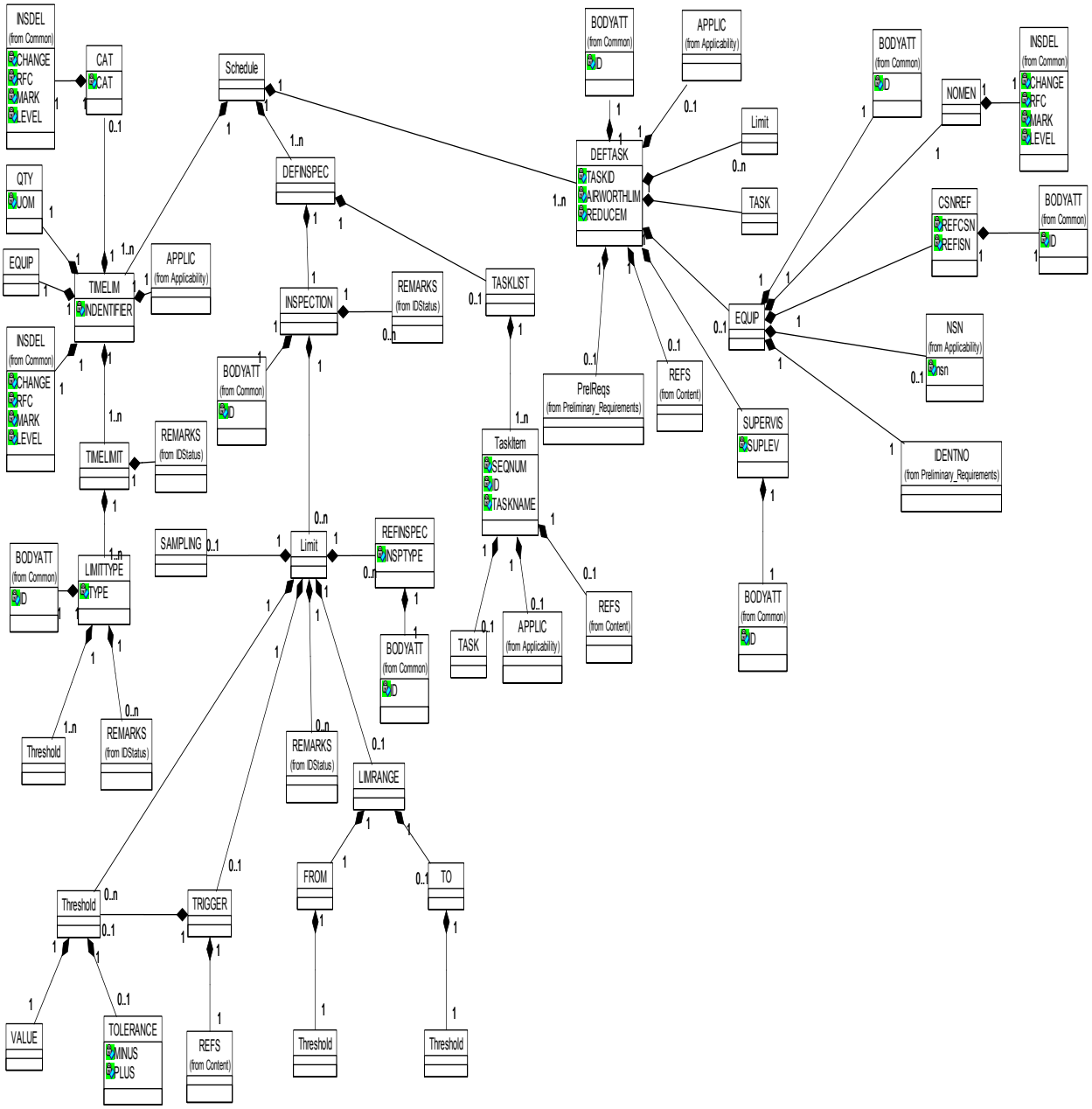


Figure A.18: Maintenance Planning Elements

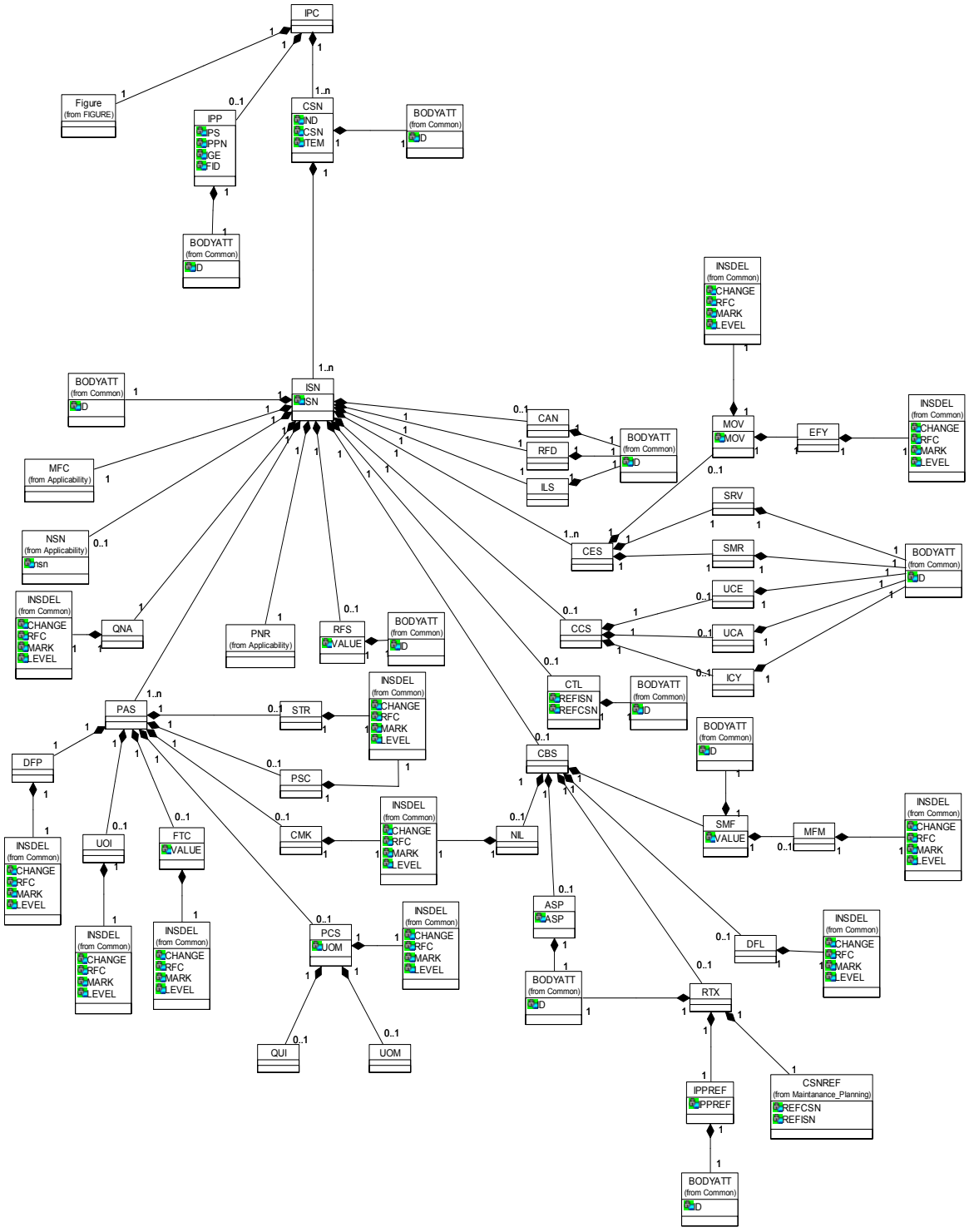


Figure A.20: Illustrated Parts Data Elements

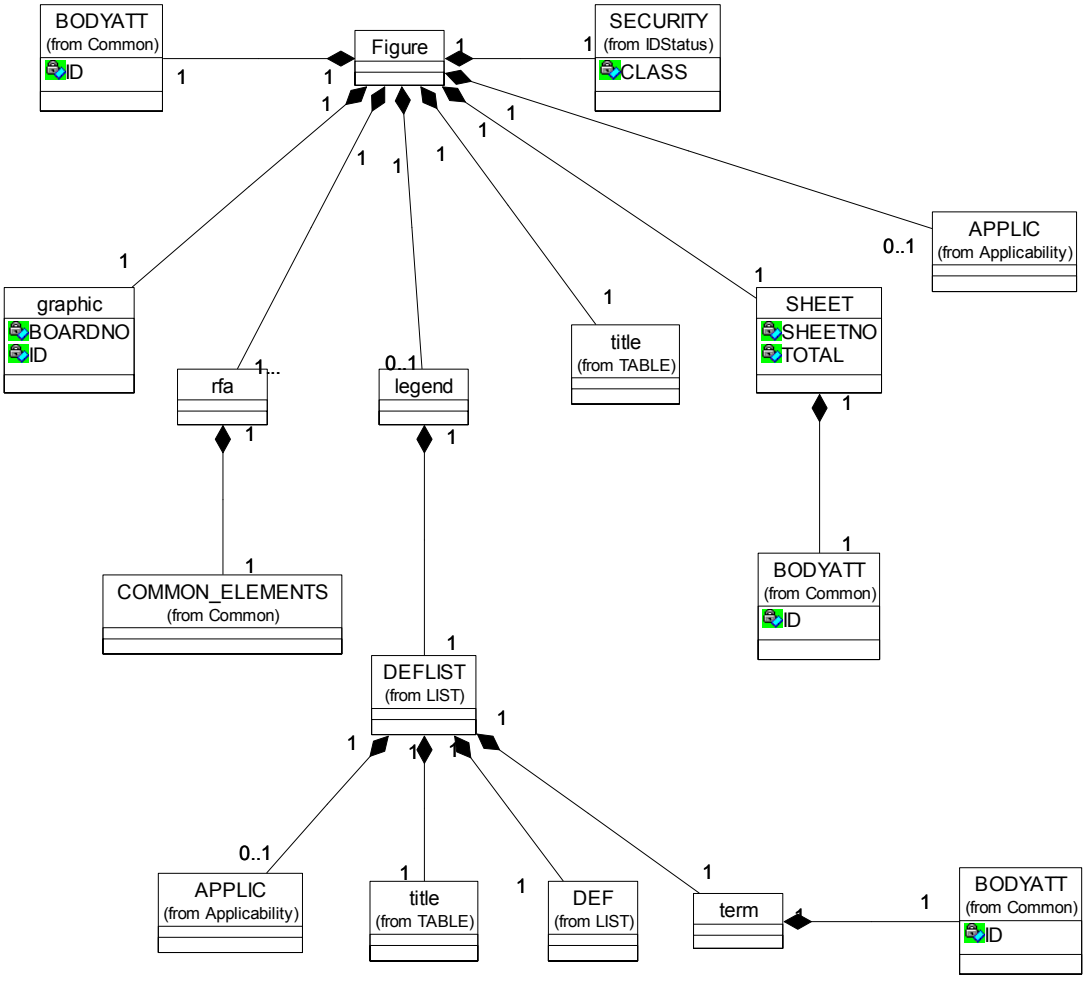


Figure A.21: Figure description Elements

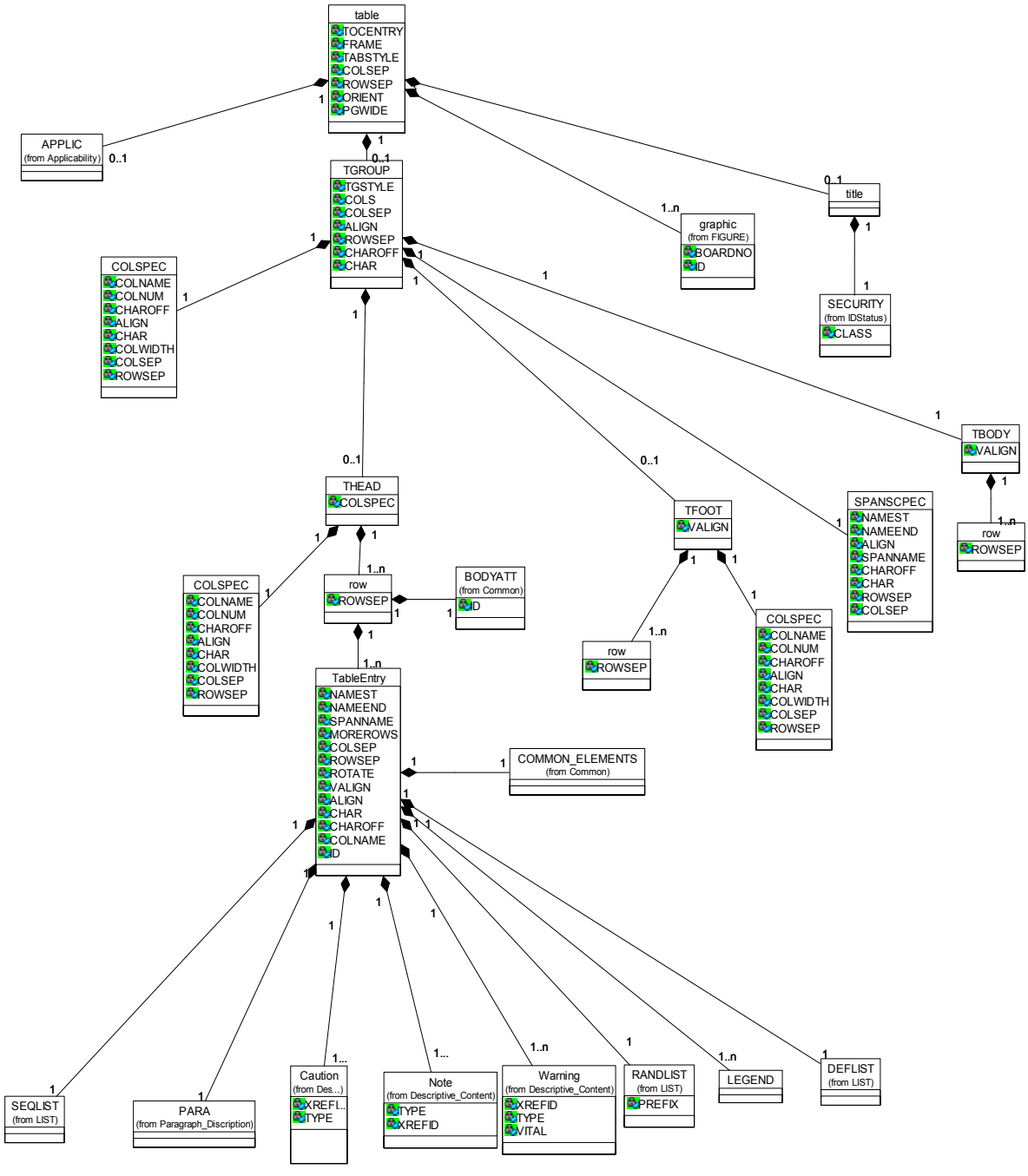


Figure A.22: Table Description Elements

Rabih F. Kraidli

OBJECTIVE

Looking for a challenging career opportunity in software development.

SUMMARY OF QUALIFICATIONS

- Strong analytical skills and innovative problem solving talents.
- Effective communication skills accomplished through presentations, meetings, and technical writings.
- Excellent teamwork skills demonstrated by responsible leadership and practical understanding of independence and interdependence between roles of team members.
- Young, ambitious and willing to put extra hours for experience
- Professional software development skills using UML, C++, JAVA, ASP, .Net and more.

EDUCATION

2001-present West Virginia University Morgantown, WV

M.S in Computer Science (GPA 4.0)

- M.S Thesis: "Web Services based Adaptive Fault Diagnosis in Interactive Electronic Technical Manuals" – Implementation in the .Net Framework.

Expected date of graduation: *May 2003*

1997-2001 Lebanese American University Beirut, Lebanon

B.S in Computer Science (GPA 3.2)

- B.S Project: "Web Enabled Environment for Technical Conference Management"

EXPERIENCE

2001–present *West Virginia University* Morgantown, WV

Graduate Research Assistant, ***Dept. of Computer Science & Electrical Eng.***

- Developed a framework for architectural-level Metrics Measurement for software (NASA IV&V sponsored project).
- Developed a framework for adaptive fault diagnosis in Interactive Electronic technical manuals (Software Engineering Research Center (SERC), ManTech International, Inc cosponsored project)
- Developed Quality Measurement Platform for error propagation in GSM cellular networks (Software Engineering Research Center (SERC), Motorola Inc. cosponsored project)

1999 - 2001 *Beirut Campaign Corp* Beirut, Lebanon

Database Administrator

- Managed Data Bases, for campaign camps for candidates running an election campaign
- Developed a voter-tracking system in election campaigns in JAVA/Perl

1997-1999 *Star Book Store* Beirut, Lebanon

Software Developer

- Developed a book archiving database system in ASP/VB

PROFESSIONAL MEMBERSHIPS

- Member, IEEE
- Member, IEEE Computer Society
- Member, Association of Computing Machinery (ACM)

TECHNICAL COURSES COMPLETED

- Interconnecting Cisco Network Devices (ICND) - Cisco certified course
- Managing Cisco Network Security (MCNS) – Cisco certified course

COMPUTER SKILLS

- Languages and Software: C++, Java, Perl, ASP, HTML, JavaScript, XML, XSLT, SML, UML, Rational Rose (Enterprise, RT).
- .Net Framework: C#, ASP.Net, VB.Net.

PUBLICATIONS

Available upon request.

REFERENCES

Available upon request.